Soft Computing (2023) 27:16959–16981
https://doi.org/10.1007/s00500-023-09024-9

OPTIMIZATION

Knowledge-based optimization algorithm for the inventory routing problem

Krzysztof Michalak[1] · **Piotr Lipinski**[2]

Accepted: 16 July 2023 / Published online: 22 August 2023

Abstract
The Inventory Routing Problem (IRP) is a combinatorial optimization problem that combines routing decisions with inventory management. In this paper, an approach to solving the IRP is studied, which aims at using an external knowledge source (a known good solution or user interaction) to improve the results attained by an evolutionary algorithm solving an IRP instance. The proposed method improves the best solution found by the evolutionary algorithm by modifying schedules for some of the retailers according to those present in the known good solution or to schedules provided by a domain expert. The experiments shown that to improve the optimization results it suffices to perform a few repetitions of the knowledge import procedure. This observation motivates further research on user-interactive optimization algorithms for the IRP, because the number of interactions needed to improve the results can easily be handled by the user.

Keywords Combinatorial optimization · Knowledge-based optimization · Transportation optimization

1 Introduction

The Inventory Routing Problem (IRP) (Bertazzi and Speranza 2012; Dantzig and Ramser 1959) is an extension of popular transportation problems on graphs, such as the Vehicle Routing Problem (VRP) (Laporte 2009), combining routing optimization with inventory management optimization (Aghezzaf et al. 2006; Archetti et al. 2007). The range of applications includes perishable goods delivery (Alkaabneh et al. 2020; Mirzaei and Seifi 2015), fuel delivery (Popović et al. 2012), maritime transport (De et al. 2017) and transportation of hazardous items (Timajchi et al. 2019). Many formulations and extensions of the regular IRP were recently studied in the literature (Archetti and Ljubić 2022), but the most common version of the IRP focuses on planning the distribution of a single product provided by a single supplier to a given number of retailers. The supplier produces a given, constant quantity of the product each day, and the retailers sell varying quantities of this product. Limited storage space is available both at the supplier and the retailers. Storage costs are calculated per unit of the product per day at a rate varying from location to location. The objective in the IRP is to optimize the inventory and transportation costs under a number of constraints on the capacity of a fleet of vehicles delivering goods, costs and limits of local inventories, etc. A solution to the IRP is a delivery schedule for a planning horizon of T days along with routes for vehicles delivering the product.

Some recent extensions concern continuous time issues. The paper by Lagos et al. (2022) proposes an approach based on dynamic discretization of the continuous time. In Lagos et al. (2020), a continuous time version of the IRP is considered. It extends the IRP with evaluating the retailer's storage capacity and product inventory at the time of the delivery. This continuous-time IRP is solved using simple time discretizations in combination with integer programming models. In Touzout et al. (2022), the Time-Dependent IRP (TD-IRP) was introduced that extends the routing part of the problem by making the travel time between two locations depend on the departure time instead of being constant as in the regular IRP. In Skalnes et al. (2022), time varying demands were investigated and a branch-and-cut algorithm based on a new mathematical formulation was introduced.

✉ Krzysztof Michalak
krzysztof.michalak@ue.wroc.pl

Piotr Lipinski
piotr.lipinski@cs.uni.wroc.pl

[1] Department of Information Technologies, Wroclaw University of Economics and Business, Wrocław, Poland

[2] Computational Intelligence Research Group, Institute of Computer Science, University of Wrocław, Wrocław, Poland

In Agra et al. (2022), a continuous time version of the IRP was studied with a single vehicle responsible for the transport from a set of suppliers to a set of retailers and two models: a location-event model and a vehicle-event model were proposed. Even though, in most cases, a finite planning horizon is considered, an infinite-horizon IRP has been formulated as early as 1985 (Blumenfeld et al. 1985; Burns et al. 1985) as an economic order quantity (EOQ) model.

Another group of extensions concerns closed-loop versions of the regular IRP. The paper by De and Giri (2020) addresses a few distinct environmental policies for carbon emission regulations and proposes an approach based on mixed integer nonlinear programming. The paper by Dev et al. (2017) studies inventory and production planning in a closed-loop system concerning both manufacturing and remanufacturing. It investigates a few inventory and production planning models and a discrete event simulation.

In Yu et al. (2022), a new variant of IRP, namely Multi-Vehicle Cyclic Inventory Routing Problem (MV-CIRP), was introduced. It tries to find the subset of retailers, the number of vehicles used, and the corresponding cycle time and route sequence in order to minimize the total cost. The paper by Yu et al. (2022) proposes a mixed-integer nonlinear programming model and a Simulated Annealing algorithm to solve it.

In Sifaleras and Konstantaras (2020), several generalizations of the IRP were discussed, including the multi-product case and the case with capacity constraints. This paper presents solving the IRP in the context of Variable Neighborhood Search (VNS).

Many formulations of the IRP are deterministic, but the version of the problem with non-deterministic lead times (Roldán et al. 2017) as well as the variant with stochastic demands (Bertazzi et al. 2013) have also been studied. In Mousavi et al. (2022), a stochastic production routing problem was introduced and discussed in the context of perishable products. This paper also addresses the uncertain demand and aims to optimize also the costs of wasted products as well as introduces penalties for non-fresh products. For solving such a problem, a matheuristic algorithm was proposed.

Some other extensions of the IRP address also returnable transport items (Iassinovskaia et al. 2017), where the supplier is also in charge of collecting the empty returnable transport items for reuse in the next cycles, as well as storage replenishment routing problem (Çelik et al. 2022), where storage replenishment operations require the transportation of items to given item slots in the storage area. Moreover, Malladi and Sowlati (2018) discuss sustainability aspects of the IRP.

Besides the extensions of the regular IRP aiming at optimizing a single objective function, some multi-objective approaches were also proposed. The paper by Rabbani et al. (2021) introduces a multi-objective optimization approach for solid waste management, focusing on recycling and waste-to-energy technologies, where the objective functions try to minimize the total net cost, greenhouse gas emissions, and total waste collection and treatment time.

Recent approaches to the IRP found in the literature are based on various algorithmic methods. A number of them have been based on integer programming and methods such as the branch-and-cut algorithm (Archetti et al. 2007). Some of them use heuristics and metaheuristics (Bard and Nananukul 2009; Hiassat et al. 2017; Lipinski and Michalak 2018; Wu et al. 2021) or iterative metaheuristics (Vadseth et al. 2021). The paper by Mahjoob et al. (2022) uses genetic algorithms (GA) for solving a multi-product, multi-period inventory routing problem. Hybrid methods have also been proposed, such as (Diabat et al. 2017) combining simulated annealing with direct search or (Liu and Chen 2011) with tabu search. Hybrid algorithms combining simulation and heuristics were proposed in Juan et al. (2014) and Juan et al. (2015).

In this paper, the approach used to solve the IRP is a metaheuristic optimization algorithm optimizing the delivery schedules. Key points summarizing the contents of this paper are:

- The paper studies the possibility of importing knowledge from an external source in order to improve the optimization results. Knowledge can be imported from a known good solution (for example, when a certain organization of deliveries is established at a company) or by interacting with the user (if a practitioner in the field wants to interactively improve the solutions). In the context of this study, knowledge imported into the optimization process is represented as schedules for individual retailers, which are either copied from a known good solution or are provided by the user.
- A hybrid optimization algorithm is used which combines evolutionary optimization of delivery schedules with optimization of routes using the Concorde TSP Solver (William 2020).
- Feasibility-preserving genetic operators (crossover, mutation) are used, which ensure that newly generated solutions are feasible and thus the population always contains feasible solutions only.
- Apart from well-known benchmarks proposed in the paper by Archetti et al. (2007), hard IRP instances evolved in the paper by Michalak (2021b) are used. These hard instances were obtained by running an evolutionary algorithm which evolved IRP *instances* with the goal of making these instances harder, that is, maximizing the solving time obtained using state-of-the-art mathematical problem solvers such as CPLEX. The solving times increased, with respect to the instances from the paper by Archetti et al. (2007), from 65.22 up to 100327.92 times (depending on the IRP instance solved) (Michalak

2021b). Therefore, these hard instances can be considered difficult to solve using state-of-the-art mathematical problem solvers.
- The evolutionary algorithm using knowledge import proposed in this paper is not likely to deteriorate the results (a worse result was only observed in 4 cases out of 720). In about 50% of tested cases, the knowledge import improved the optimization results and in the remaining cases the results were not statistically different for the algorithm with and without the knowledge import.
- For the instances proposed in the paper by Archetti et al. (2007) the knowledge import mechanism improved the results in about 66% of cases in a scenario with only one knowledge import, in which 10 changes of the schedules for individual retailers were attempted. For the *Hard* instances evolved in the paper by Michalak (2021b) the knowledge import improved the results in about 16% of cases and never made the results worse. This means, that if the user can be expected to provide 10 good schedules for individual retailers in just one interactive session during the entire algorithm's runtime, the optimization results can be improved without the risk of degrading the performance of the proposed method. Therefore, the workload of the user should not be too high, and the proposed approach appears to be suitable for designing user-interactive optimization methods for the IRP.
- For the *Hard* instances evolved in the paper by Michalak (2021b) the knowledge-based EA presented in this paper attained the best results among all the tested methods within the running time limit of $\max_t = 3600$ s (1 h). This shows that the proposed method can be effectively used for solving hard IRP instances for which mathematical programming methods require a very long running time, and known metaheuristic methods are not competitive to the one proposed here.

The rest of this paper is structured as follows. Section 2 presents the formulation of the IRP studied in this paper. Section 3 discusses the optimization algorithms used in this paper to solve the IRP. Section 4 presents the experiments and results. Section 5 concludes the paper.

2 The inventory routing problem formulation

In this paper, a single-depot, single-commodity and single-vehicle version of the IRP with the up-to-level replenishment policy presented by Archetti et al. (2007) is studied. A single commodity is delivered from one supplier (depot) to n retailers with one vehicle with capacity C. The deliveries have to be planned within a horizon of H consecutive time periods (days). Variables $r_{s,t}$ for $s = 0, \ldots, n, t = 1, \ldots, H$

$$\begin{array}{lccccc} t=1: & 1 & 2 & 7 & 5 & 3 \\ t=2: & 6 & 7 & 8 & 4 & 9 \\ t=3: & 5 & 7 & & & \end{array}$$

Fig. 1 The optimal solution to the hard3n10 IRP instance generated in the paper by Michalak (2021b)

represent the production at the supplier and sales at the retailers. The index $s = 0$ is assigned to the supplier, so $r_{0,t}$ is the production and $r_{s,t}$ for $s > 0$ are the sales on day t. Inventory levels at the supplier are represented by variables B_t, $t = 1, \ldots, H$. There is no upper limit on B_t, which means that the supplier always has enough storage capacity for the produced goods, and the lower bound is $\forall t = 1, \ldots, H : B_t \geq 0$, which means that stock-out at the supplier is not allowed. Inventory levels at the retailers are represented by variables $I_{s,t}$ and the minimum and maximum inventory levels denoted L_s and U_s are constant in time, but potentially different for every retailer. Therefore, the inventory level constraints are represented by inequalities:

$$\forall s = 1, \ldots, n, \forall t = 1, \ldots, H : L_s \leq I_{s,t} \leq U_s \ . \tag{1}$$

Following the work of Archetti et al. (2007) we set $\forall s = 1, \ldots, n : L_s = 0$, which means that no minimum inventory levels are required at the retailers, except for the *no stock-out* requirement. A solution to the IRP consists of the routes for the vehicle π_t and the delivery sizes $x_{s,t}$ for $s = 1, \ldots, n, t = 1, \ldots, H$. Each route π_t is a permutation of a subset of the n retailers— those, which are visited on day t. Figure 1 shows the optimal solution to the hard3n10 IRP instance generated in the paper by Michalak (2021b).

The tour of the vehicle always starts and ends at the supplier, so the supplier does not have to be explicitly included in π_t, but the transportation cost (6) includes the costs of driving from the supplier to the first retailer in π_t and from the last retailer in π_t back to the supplier. In this paper, the *up-to-level* replenishment policy is used, so only the routes (permutations) π_t are used to represent solutions and the delivery sizes are calculated as:

$$x_{s,t} = \begin{cases} U_s - I_{s,t} & \text{if} \quad s \in \pi_t \\ 0 & \text{if} \quad s \notin \pi_t \end{cases} , \tag{2}$$

The maximum vehicle capacity C cannot be exceeded by all the goods delivered on each day:

$$\forall t = 1, \ldots, H : \sum_{s=1}^{n} x_{s,t} \leq C \ . \tag{3}$$

Location-specific inventory costs h_s, $s = 0, \ldots, n$, are assumed to be constant in time and the overall storage costs

are the costs of storing the commodity until the next planning horizon arrives:

$$\sum_{t=1}^{H+1} h_0 B_t + \sum_{s=1}^{n} \sum_{t=1}^{H+1} h_s I_{s,t} \ . \tag{4}$$

Transportation costs are calculated using costs $c_{i,j}$ of traveling between each pair of points $i, j = 0, \ldots, n$. In this paper, these costs are calculated as rounded Euclidean distances between locations given as coordinates $a_s, b_s \in \mathbb{R}^2$, $s = 0, \ldots, n$:

$$c_{i,j} = \left[\sqrt{(a_j - a_i)^2 + (b_j - b_i)^2} \right] \ , \tag{5}$$

where $[\cdot]$ denotes the operation of rounding a given number to the nearest integer. The cost $c(\pi_t)$ of completing a given route $\pi_t = [\pi_{t,1}, \ldots, \pi_{t,k}]$ is:

$$c(\pi_t) = c_{0,\pi_{t,1}} + \sum_{i=1}^{k-1} c_{\pi_{t,i},\pi_{t,i+1}} + c_{\pi_{t,k},0} \ . \tag{6}$$

As mentioned above, when the *up-to-level* replenishment policy is used, a solution to the IRP is fully described by a sequence of routes $\{\pi_t\}_{t=1,\ldots,H}$. To evaluate such solution, the deliveries $x_{s,t}$ for each day have to be calculated using Eq. (2). Given the initial inventory levels $I_{s',0}$, deliveries $x_{s,t}$, and production/consumptions $r_{s,t}$, the inventory levels B_t and $I_{s,t}$ are determined. The evaluation of the solution is obtained by adding the costs of all routes π_t, $t = 1, \ldots, H$ calculated using Eq. (6) and the inventory costs calculated using Eq. (4).

In order not to clutter the paper, for the mathematical formulation suitable for MIP solvers readers are referred to Eqs. (1–16) in the work of Archetti et al. (2007).

Denoting $\mathcal{M} = 1, \ldots, n$, $\mathcal{M}' = \mathcal{M} \cup \{0\}$ and assuming that the production and sales are constant in time: $\forall s' = 0, \ldots, n, \forall t : r_{s',t} = r'_s$ an instance of the IRP considered in this paper can be defined as a tuple of the following values:

$$\mathcal{I} = \langle n, H, C, a_{s'}, b_{s'}, r_{s'}, L_s, U_s, I_{s',0}, h_{s'} \rangle_{s \in \mathcal{M}, s' \in \mathcal{M}'} \ , \tag{7}$$

where n—the number of retailers, H—the planning horizon, C—the vehicle capacity, $a_{s'}, b_{s'}$—the coordinates of the locations (the supplier and the retailers), $r_{s'}$—the production (at the supplier for $s' = 0$) and sales (at the retailers for $s' > 0$), L_s—the lower inventory levels at the retailers, U_s—the upper inventory levels at the retailers, $I_{s',0}$—the initial inventory levels at the supplier and retailers, $h_{s'}$—location-specific inventory costs at the supplier and retailers.

The IRP is an NP-hard problem, because it is a generalization of the Traveling Salesperson Problem (TSP), which is obtained from the IRP by setting $H = 1$, $C = \infty$, $I_{0,0} \geq n$, $I_{s,0} = 0$ and $r_s = 1$ for $s = 1, \ldots, n$, and $h_s = 0$ for $s = 0, \ldots, n$. For these settings, only the transportation costs are considered and the vehicle has to visit all the retailers (and thus also the supplier) on the same day.

In the discussion in subsequent sections the following terminology will be used:

A *schedule* is a sequence of days on which a given retailer is visited. For example, for the solution shown in Fig. 1 the schedule for retailer 1 is [1], for retailer 5 it is [1, 3], and for retailer 7 it is [1, 2, 3].

A *route* is a sequence of retailers visited on a given day. For example, for the solution shown in Fig. 1 the route for day 1 is [1, 2, 7, 5, 3].

A *solution* consists of H routes, one per each day within the planning horizon. The optimal solution to the hard3n10 IRP instance (with $H = 3$) generated in the paper by Michalak (2021b) is shown in Fig. 1.

3 Knowledge-based optimization for the IRP

The optimization algorithm proposed in this paper is a hybrid algorithm combining an evolutionary algorithm (EA) with the Concorde TSP Solver (William 2020) which optimizes the routes. The evolutionary algorithm is based on the classical genetic algorithm scheme with a recursive heuristic used for initializing the population and feasibility-preserving genetic operators. This algorithm involves a knowledge import step, which improves the best solution found by the EA by user interaction or by importing a known good delivery schedule for one of the retailers. In the following sections, the details of the proposed method are presented. An introductory Sect. 3.1 lists procedures used in the proposed method. In Sect. 3.2 the heuristic used for obtaining the base solution for population initialization is described. Section 3.3 discusses the details of the optimization algorithm. Section 3.4 presents the mechanism for importing knowledge into the optimization process.

3.1 Procedures used in the proposed method

The optimization algorithm used in this paper is an evolutionary algorithm (Algorithm 4) based on the classical genetic algorithm scheme. In this algorithm, the *SolveTSP* procedure is used for optimizing the routes in each solution using the Concorde TSP Solver (William 2020). The *ImportKnowledge* procedure (Algorithm 5) improves the best solution found by the EA by user interaction or by importing information from a known good solution.

Procedures used for population initialization are:

BaseSolution (Algorithm 1)— the recursive heuristic for generating the base solution.

AddRetailerToSolution (Algorithm 2) called recursively n times in the BaseSolution procedure to add schedules for n retailers in a random order to the base solution.

GeneratePossibleSchedules (Algorithm 3) used in the AddRetailerToSolution procedure for recursively generating all possible delivery schedules for a given retailer which do not cause the vehicle to be overloaded.

FeasibleDateChangeMutation—a mutation operator proposed in the paper by Michalak (2021a), which moves retailers between different days, ensuring that the solution remains feasible.

Procedures used by the main loop of the evolutionary algorithm are:

Evaluate—evaluates a given solution by adding the costs of all routes π_t, $t = 1, \ldots, H$ calculated using equation (6) and the inventory costs calculated using equation (4). After this procedure is called for a solution x, the evaluation (the total cost) of this solution can be obtained by referring to the attribute $x.Eval$.

Reduce—reduces the union of the current population (N_{pop} solutions) and the offspring generated by the genetic operators (N_{pop} new solutions) back to the required population size $N_{\text{pop}} - 1$ (leaving room for a copy of the best specimen).

Reproduce—applies the crossover and mutation operators to obtain the next population from the current one.

CopyBestSpecimen—creates a copy of the best specimen from a given population (i.e. the solution with the lowest cost).

FindWorstSpecimen—finds the index in the population at which the worst specimen can be found (i.e. the solution with the highest cost).

Procedures used for manipulating schedules for individual retailers are:

AddRetailerDays—adds the specified retailer s to the solution x on days given in a schedule S. For example if x is the solution presented in Fig. 1, then AddRetailerDays(x, 2, [2, 3]) produces a solution:

$t = 1$:	1	2	7	5	3	
$t = 2$:	6	7	8	4	9	2
$t = 3$:	5	7	2			

Note, that the routes with the added retailer are not optimized until the SolveTSP procedure is called for the solution x.

GetKnownDays—imports knowledge from an external source by either copying a schedule for a certain retailer s^* from a known good solution or by consulting the user and asking for a proposed schedule for that retailer. Either way, a schedule for the retailer s^* (a sequence of days on which this retailer should be visited) is returned.

Algorithm 1: The BaseSolution procedure - the recursive heuristic for generating the base solution used for initializing the population in the evolutionary algorithm.

Inputs:
$\mathcal{I}$ - An IRP instance defined as in the equation (7).

Output:
The base solution generated by this heuristic.

// Start with an empty solution - H empty routes.
$x := \{[], [], \ldots, []\}$

// Vehicle loads for the H days are initially zero.
$V := [0, 0, \ldots, 0]$

// A random permutation of n retailers.
$\pi :=$ RandPerm($\mathcal{I}.n$)

// Call the AddRetailerToSolution procedure (Algorithm 2) which recursively builds the solution.
$x_{base} :=$ AddRetailerToSolution($\mathcal{I}$, x, V, π, 0)

// Return the generated base solution.
return x_{base}

GetRetailerDays—returns the days on which a given retailer is visited in a given solution. For example, if x is the solution presented in Fig. 1, then GetRetailerDays(x, 5) returns [1, 3].

SortSchedules—sorts schedules generated for a retailer by the GeneratePossibleSchedules procedure (Algorithm 3), placing shorter schedules (with fewer delivery days) first. Schedules of equal length (the same number of delivery days) are sorted by placing those with earlier delivery days first.

SupplyAtTheLatestDate—generates a schedule for a retailer using the Supply at the Latest Date (SLD) heuristic previously used in the papers (Lipinski and Michalak 2018, 2019; Michalak 2021a), which determines the latest possible days at which the deliveries have to be made to a given retailer in order to avoid stockout.

If processing elements of a given set in a random order is necessary, the *RandPerm* procedure is used, which generates a random permutation of a given length.

3.2 Recursive heuristic for generating the base solution

The recursive heuristic is used to obtain the base solution which is used in the evolutionary algorithm to initialize the population. The initial population consists of the base solution and its mutated copies. The recursive heuristic implemented in the BaseSolution procedure (Algorithm 1) generates the base solution by adding schedules for n retailers

Algorithm 2: The AddRetailerToSolution procedure.

Inputs:
- $\mathcal{I}$ - An IRP instance defined as in the equation (7).
- x - A partial solution (a sequence of H incomplete routes).
- V - Vehicle loads on H days of the planning horizon calculated for the incomplete routes in x.
- π - A random permutation which determines the order in which to add retailers to the solution.
- i - The retailer to add (an index used to select an element from π).

Output:
A full solution (a sequence of H complete routes).

```
// The retailer to add selected from the permutation π
s := π[i]

// The day on which the initial inventory runs out for retailer s.
d_0 := ⌊ (I.I_{s,0} − I.L_s) / I.r_s ⌋ + 1;

// Deliveries are only needed if the initial inventory is not enough.
if d_0 < H then
    // Generate all possible delivery schedules for the retailer s.
    // Each schedule is a set of days on which the retailer s is visited.
    J := GeneratePossibleSchedules(I, s, ∅, 1, V, I_{0,0}, I_{s,0})
    if |J| = 0 then
        return null

    // Sort the schedules, placing shorter schedules (with fewer delivery days) first.
    // Schedules of equal length are sorted by placing those with earlier delivery days first.
    SortSchedules(J)

    if i < I.n then
        for j := 1, ..., |J| do                          // Try the schedules in the sorted order.
            x' := x; V' := V; q := I_{s,0}
            for t := 1, ..., H do
                if t ∈ J[j] then
                    x'[t] := x'[t] ∪ {s}                 // Add the retailer s to the route for day t.
                    q' := U_s − q                        // The quantity delivered on day t.
                    q := q + q'                          // The inventory level set to U_s.
                    V'[t] := V'[t] + q'                  // Increase the vehicle load on day t. Note, that schedules
                                                         // generated by the GeneratePossibleSchedules procedure
                                                         // never exceed the vehicle capacity.
                q := q − r_s

            // Recurse to add the next retailer.
            x'' := AddRetailerToSolution(I, x', V', π, i + 1)
            if x'' ≠ null then
                return x''
    else
        // Retailer s is the last one - just add the first schedule generated for s.
        for t ∈ J[1] do                                  // For each day t in the schedule J[1]
            x[t] := x[t] ∪ {s}                           // add the retailer s to the route for day t.
        return x
else
    // No deliveries needed...
    if i < I.n then
        // ... and we can proceed to the next retailer.
        return AddRetailerToSolution(I, x, V, π, i + 1)
    else
        // ... and s is the last retailer - the solution is complete.
        return x
```

in a random order using n recursive calls to the AddRetailerToSolution procedure (Algorithm 2) which is started with an empty solution and a random permutation which determines the order in which schedules for retailers are added. In the AddRetailerToSolution procedure the GeneratePossibleSchedules procedure (Algorithm 3) is used for recursively generating all possible delivery schedules for a given retailer which do not cause the vehicle to be overloaded.

3.3 Optimization algorithm

In this paper, a hybrid optimization algorithm is used which combines evolutionary optimization of delivery schedules with optimization of routes using the Concorde TSP Solver (William 2020). The algorithm uses feasible crossover and mutation operators described by Michalak (2021a) which ensure that only feasible solutions are generated. The genetic

Algorithm 3: The GeneratePossibleSchedules procedure.

Inputs:

$\mathcal{I}$	-	An IRP instance defined as in the equation (7).
s	-	The retailer for which to generate schedules.
D	-	A partial schedule for retailer s (days assigned to s so far).
t	-	The day which is considered to be added to D. When $t > 1$, days $1, \ldots, t-1$ were already considered for addition to D.
V	-	Vehicle loads on H days of the planning horizon.
i_0	-	The inventory level at the supplier at the beginning of day t.
i_s	-	The inventory level at the retailer s at the beginning of day t.

Output:

A set of generated schedules for retailer s.
Each schedule is a set of days on which the retailer s is visited.

```
// If t is the next day after the planning horizon then D is a full schedule for retailer s.
// This schedule can be added to the set of possible schedules if no stockout at the supplier occurs.
if t = I.H + 1 then
    if i_0 ≥ 0 then
        return {D}
    else
        return ∅

// If stockout at the supplier occurs at day t then D is not a valid partial schedule.
if i_0 < V[t] then
    return ∅

// Possible schedules will be added recursively to J.
J := ∅

// Schedules without day t can be added if there is no stockout at retailer s on day t.
if i_s ≥ I.r_s + I.L_s then
    J := J ∪ GeneratePossibleSchedules(I, s, D, t + 1, V, i_0 + I.r_0 − V[t], i_s − I.r_s)

// Schedules with day t can be added if there is no stockout at the supplier and vehicle capacity is not exceeded on day t.
q' := I.U_s − i_s                    // The quantity required to make the inventory full at retailer s.
if (V[t] + q' ≤ i_0) ∧ (V[t] + q' ≤ I.C) then
    J := J ∪ GeneratePossibleSchedules(I, s, D ∪ {t}, t + 1, V, i_0 + I.r_0 − V[t] − q', i_s − I.r_s + q')

// The result is the set of schedules generated recursively with and without day t.
return J
```

operators used in this paper are the following (for details readers are referred to the paper by Michalak (2021a)):

Feasible Retailer Schedules Crossover which works by uniformly mixing delivery schedules for individual retailers obtained from two parent solutions.

Feasible Date Change Mutation which considers retailers in a random order selecting, for each retailer, one date when it is visited, and moving this retailer to some other feasible date.

Feasible Add Or Remove Retailer Mutation which randomly applies one feasible change to the mutated solution: an addition or removal of a retailer.

Because the population initialization procedure generates only feasible solutions and genetic operators preserve the feasibility, no mechanisms for handling infeasible solutions, such as repair operators, are needed in the proposed algorithm. The most important characteristics of the algorithm used in this paper are:

- Population initialized by generating the base solution using a recursive heuristic described in Sect. 3.2 which is subsequently mutated using the *Feasible Date Change Mutation* operator. Such population initialization method ensures that the initial population consists only of feasible solutions.
- Feasibility-preserving genetic operators (crossover, mutation), which ensure that newly generated solutions

Algorithm 4: The optimization algorithm used in this paper.

Inputs:
- $\mathcal{I}$ - An IRP instance defined as in the equation (7).
- N_{pop} - Population size.
- $\max_{FE}$ - Stopping condition: the maximum number of solution evaluations.
- N_{imp} - The number of knowledge imports to perform during the algorithm's run.
- N_{att} - The number of attempts to improve the solution in each knowledge import.

Output:
- x_{best} - The best solution found by the algorithm.

```
// Generate the base solution.
x_base := BaseSolution(I)
SolveTSP(x_base)
Evaluate(x_base);   N_eval := 1

// Initialize the population using the base solution
// and its mutated copies.
P := {x_base}
for p := 1, ..., N_pop − 1 do
    x := FeasibleDateChangeMutation(x_base)
    SolveTSP(x)
    Evaluate(x);   N_eval := N_eval + 1
    P := P ∪ {x}
x_best := CopyBestSpecimen(P)

// The main loop
g := 1
imp := 1
do
    // Knowledge import performed at equal intervals
    if N_eval ≥ imp/(N_imp+1) max_FE then
        x_imp := ImportKnowledge(I, x_best, N_att)
        if x_imp.Eval < x_best.Eval then
            i_worst := FindWorstSpecimen(P)
            P[i_worst] := x_imp
        imp := imp + 1

    // Reproduction (crossover, mutation)
    P' := Reproduce(P)
    for x ∈ P' do
        SolveTSP(x)
        Evaluate(x);   N_eval := N_eval + 1

    // Reduction of the population size
    P := Reduce(P ∪ P', N_pop − 1)
    P := P ∪ {x_best}
    x_best := CopyBestSpecimen(P)

    // Next generation
    g := g + 1
while N_eval < max_FE
x_best := CopyBestSpecimen(P)
return x_best
```

Algorithm 5: The ImportKnowledge procedure.

Inputs:
- $\mathcal{I}$ - An IRP instance defined as in the equation (7).
- x_0 - The solution to improve.
- N_{att} - The number of attempts to improve the solution.

Output:
- x^* - An improved solution.

```
// The best improved solution found by the algorithm.
x* := x_0

// A random permutation used to select retailers
// without repetitions.
π* := RandPerm(I.n)

// Take each retailer in turn as the one for which
// the schedule is taken from the knowledge source.
for i := 1, ..., N_att do
    s* := π*[i]

    // Start with an empty solution - H empty routes.
    x := {[], [], ..., []}

    // First, add the schedule for retailer s* from
    // the knowledge source.
    S := GetKnownDays(s*)
    AddRetailerDays(x, s*, S)

    // Add the schedules for other retailers from x_0
    // in a random order.
    π := RandPerm(I.n)
    for j := 1, ..., I.n do
        s := π[j]
        if s ≠ s* then
            S := GetRetailerDays(x_0, s)
            AddRetailerDays(x, s, S)

    SolveTSP(x)
    Evaluate(x)
    if x.Eval < x*.Eval then
        x* := x
return x*
```

are feasible and thus the population always contains only feasible solutions.
- Fitness calculated based on the solution costs normalized to [0, 1]. That is $fitness(x) = 1 - Norm(f(x), 0, 1)$, where the $Norm$ function normalizes the solution costs in the current population to [0, 1]. The evolutionary algorithm tries to maximize $fitness(x)$ thereby minimizing the cost $f(x)$.
- Elitism mechanism, preserving the best solution found so far.
- Mating pool selection based on a binary tournament with respect to the fitness.
- Hybridization with the Concorde TSP Solver (William 2020). The evolutionary algorithm optimizes delivery schedules, that is, decides which retailers are visited on

which days. The optimization of the route of the vehicle on each day is done by the Concorde solver.

The working of the optimization algorithm is shown in Algorithm 4.

3.4 Knowledge import

The knowledge import improves the best solution found so far by the evolutionary algorithm x_{best} by incorporating information from an external knowledge source, which can be user interaction or a known good solution to the solved problem instance. It is performed by changing schedules (days on which the retailers are visited) for individual retailers in the x_{best} solution to those provided by the knowledge source. The knowledge import is implemented as the ImportKnowledge procedure (Algorithm 5), which performs N_{att} attempts to improve the x_{best} solution by:

1. Selecting one of the retailers $s^* \in \mathcal{M}$ randomly without repetitions.
2. Getting a schedule for the retailer s^* from the external knowledge source. This step is performed by the GetKnownDays procedure called in Algorithm 5, which either copies the schedule for the retailer s^* from a known good solution or consults the user and asks for a proposed schedule for that retailer.
3. Copying the schedules for all the retailers in $\mathcal{M}\backslash\{s^*\}$ from the solution x_{best}. If adding a schedule from x_{best} for a certain retailer causes the solution to be infeasible the schedule for this retailer is generated using the SupplyAtTheLatestDate heuristic instead.

From the N_{att} attempts to improve the solution the best one x^* is selected (in Algorithm 5) and, if it is better than x_{best}, it replaces the *worst* solution in the EA population (in Algorithm 4). This approach is adopted in order to protect the diversity of the population. Because both x^* and x_{best} are kept, both the information generated by the evolutionary optimizer and coming from the external knowledge source can be used to generate new solutions in the evolving population. The worst solution in the population, which is replaced by x^*, would probably be removed by the selection mechanism anyway, so it can be assumed that no valuable information is lost because of this replacement.

4 Experiments and results

The experiments presented in this paper were aimed at determining if importing knowledge from an external source can be beneficial for the optimization process. It was assumed that the knowledge import can be performed using a known good solution to the problem (e.g. established by practitioners in the field) or by user interaction. In both cases the import is performed by modifying schedules for some retailers in the best solution found by the evolutionary algorithm. This modification can be performed by the user or by setting the days on which these retailers are visited to those which are assigned to these retailers in a known good solution.

In the experiments, the parameters of the evolutionary algorithm were tuned (Sect. 4.1), the running times of the evolutionary algorithm solving the hardest IRP instance obtained in the paper by Michalak (2021b) were analyzed and compared to the running times of the CPLEX solver (Sect. 4.2), and the effectiveness of the proposed knowledge import method was tested (Sect. 4.3). In Sect. 4.4 the proposed method was compared to metaheuristics used in the literature to solve the IRP and to the mathematical programming model proposed in the paper by Archetti et al. (2007) in which the *Archetti 2007 LC* and *Archetti 2007 HC* IRP instances were provided.

4.1 Parameter tuning

In this paper, an evolutionary algorithm described in Sect. 3 is used for solving the IRP. This algorithm is parameterized by setting the population size N_{pop}, crossover probability P_{cross}, and mutation probability P_{mut}. In order to ensure the best performance of the optimization algorithm, these parameters were tuned using the grid search approach. The values of the parameters were selected from the sets of candidate values: $N_{\text{pop}} \in \{50, 100, 200, 500\}$, $P_{\text{cross}} \in \{0.5, 0.6, 0.7, 0.8, 0.9, 1.0\}$, $P_{\text{mut}} \in \{0.02, 0.04, 0.06, 0.08, 0.10\}$. Apart from these three numerical parameters, the *Reduce* procedure used in Algorithm 4 can work in one of four modes:

- *Elitism*—The best solutions are selected from the union $P \cup P'$ of the current population P and the offspring P'.
- *Fitness*—Solutions are selected from the union $P \cup P'$ of the current population P and the offspring P' using the roulette-wheel selection with the probabilities proportional to the fitness.
- *New*—Only the offspring P' are kept.
- *Rank*—Solutions are selected from the union $P \cup P'$ of the current population P and the offspring P' using roulette-wheel selection with the probabilities proportional to the rank calculated with respect to the increasing fitness. The solution with the worst (lowest) fitness gets a rank of 1, and the solution with the best (highest) fitness gets a rank of $|P \cup P'|$, where $|P \cup P'|$ denotes the number of solutions in the current population P and the offspring P' population taken together. In the case of two solutions with the same fitness, two consecutive ranks are assigned at random, so one solution gets a rank r and the other one $r+1$. After the ranks are assigned, the roulette-

Table 1 Parameters of the distribution of the running times observed in the parameter-tuning phase of the experiments

Parameter	Time (s)
Min	3637.0
Max	23448.0
Median	6430.5
Mean	7434.9
SD	3270.6

wheel selection is performed, in which each solution has the probability of being selected proportional to its rank.

For each reduction mode and a triple of the parameters N_{pop}, P_{cross}, and P_{mut}, 10 runs of the test were performed using the IRP instance *hard4n35* constructed in the paper by Michalak (2021b). This instance was selected because it was the hardest one found in that paper, that is, it required the longest solving time using the CPLEX solver. The parameters for which the best (lowest) average cost of 5049.338 was obtained were selected. The best settings were: the Elitism reduction mode, $N_{\text{pop}} = 200$, $P_{\text{cross}} = 0.8$, $P_{\text{mut}} = 0.1$.

4.2 Running times

The solving time using the CPLEX solver for the IRP instance *hard4n35* reported in the paper by Michalak (2021b) was 354681.28 s (more than 4 days) on a machine with the Intel Xeon E5-2670v3 (Haswell) CPUs running at 2.3 GHz. The optimal cost attained within this running time was 5043.55. During the parameter tuning phase of the experiments presented in Sect. 4.1, 480 runs of the evolutionary algorithm were performed (4 reduction modes × 4 population sizes × 6 crossover probabilities × 5 mutation rates). Parameters of the distribution of the running times observed in these 480 runs on a machine with the same specification (i.e. the Intel Xeon E5-2670v3 (Haswell) CPUs running at 2.3 GHz) are presented in Table 1. In Fig. 2 the histogram of the running times is presented. Clearly, the running times of the optimization algorithm studied in this paper are competitive to those of the CPLEX solver. The maximum running time for the EA is less than 7% of the running time for CPLEX (23448.0 s). The majority of timings is much lower than that.

4.3 Experiments with knowledge import

The knowledge import mechanism was tested using the optimization algorithm presented in Algorithm 4 with the parameters tuned in Sect. 4.1. The number of knowledge imports was set to $N_{\text{imp}} = 0$, 1, 2, 5, and 10. The number of attempts to improve the solution in each knowledge import was set to $N_{\text{att}} = 10$ and $N_{att} = \mathcal{I}.n$. The constant value $N_{\text{att}} = 10$ is a reasonable choice for a user-interactive

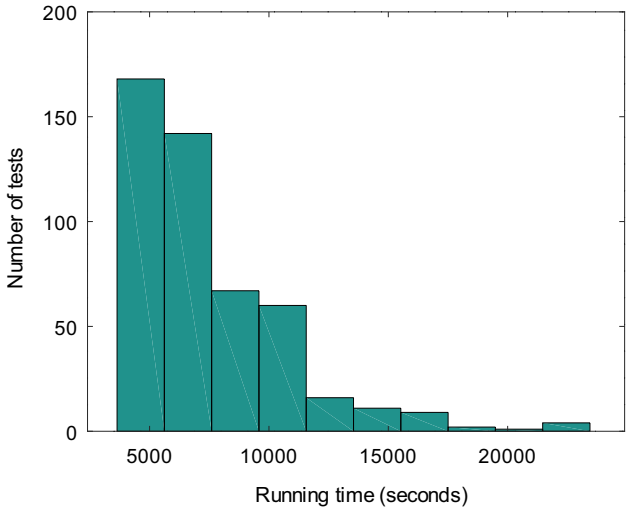

Fig. 2 The histogram of the running times observed in the parameter-tuning phase of the experiments

approach, because with such setting the user has to suggest 10 new schedules for different retailers, which is not overly time consuming. The setting $N_{\text{att}} = \mathcal{I}.n$ can be a good choice for importing knowledge from a known good solution, because it allows trying to import the schedule for each of the retailers in turn. With these settings, the knowledge import can be expected to have a small impact on the algorithm's running time, because $N_{\text{imp}} \cdot N_{\text{att}} = 350$ new solutions are imported for the maximal values of $N_{\text{imp}} = 10$ and $N_{\text{att}} = \mathcal{I}.n = 35$, which is 3.5% of the stopping criterion value of $\max_{\text{FE}} = 10000$ solution evaluations.

The experiments were performed using three sets of IRP instances:

- *Archetti 2007 LC*—IRP instances with the planning horizon $H = 6$ and low inventory costs proposed in the paper by Archetti et al. (2007). This set of IRP instances contains 5 instances for each number of retailers $n = 5$, 10, 15, 20, 25, and 30.
- *Archetti 2007 HC*—IRP instances with the planning horizon $H = 6$ and high inventory costs proposed in the paper by Archetti et al. (2007). This set of IRP instances contains 5 instances for each number of retailers $n = 5$, 10, 15, 20, 25, and 30.
- *Hard*—IRP instances with the planning horizon $H = 3$ evolved in the paper by Michalak (2021b) from the *Archetti 2007 LC* instances with the planning horizon $H = 3$. This set of IRP instances contains 5 instances for each number of retailers $n = 10$, 15, 20, 25, 30, and 35. These instances were generated using an evolutionary algorithm in such a way that the solving time with state-of-the-art mathematical problem solvers was very long. The solving times using the CPLEX solver on a machine

Table 2 A comparison of the results obtained using the knowledge import to the results obtained without the knowledge import

Dataset	Archetti 2007 LC		Archetti 2007 HC		Hard		All
N_{att}	10	$\mathcal{I}.n$	10	$\mathcal{I}.n$	10	$\mathcal{I}.n$	
Num. worse (−)	0	0	2	2	0	0	4
Num. equal (=)	39	37	43	41	100	93	353
Num. better (+)	81	83	75	77	20	27	363

with the Intel Xeon E5-2670v3 (Haswell) CPUs running at 2.3 GHz reported in the paper by Michalak (2021b) ranged from 238.93 s (for the *hard4n10* instance with $n = 10$ retailers) to 354681.28 s (for the *hard4n35* instance with $n = 35$ retailers). The solving times for the *Hard* instances increased, with respect to *Archetti 2007 LC*, instances from 65.22 up to 100327.92 times (depending on the IRP instance solved). Therefore the *Hard* instances can be considered difficult to solve using state-of-the-art mathematical problem solvers.

For each problem instance and the pair of parameters N_{imp} and N_{att} the optimization algorithm was run 200 times. In order to ensure repeatability of the experiments, the schedules for retailers were imported from optimal solutions to these problem instances found by the CPLEX solver instead of user interactions. A summary of the results is presented in Table 2. In order to test if the knowledge import affected the optimization results, the Friedman test was applied to the results obtained for each dataset and each value of N_{att} with groups ("treatments") representing different values of N_{imp}. The null hypothesis of this test is that the medians of the optimization results are equal across all groups (values of N_{imp}). The results of the Friedman test are presented in Table 3. Based on low p-values obtained in this test it can be concluded that there are statistically significant differences of optimization results among different values of N_{imp}. Detailed results are presented in Tables 4, 5, 6, 7, 8, 9 in which median values obtained in 200 runs of the optimization algorithm are shown. The column $N_{\text{imp}} = 0$ contains results obtained without using the knowledge import, and columns with $N_{\text{imp}} > 0$ contain results obtained when a given number of knowledge imports was performed. Apart from numerical results, these tables present the results of a statistical comparison performed using the Wilcoxon statistical test (Rey and Neuhäuser 2011) in which the null hypothesis states the equality of medians. The sign '(+)' is used to mark those results obtained for $N_{\text{imp}} > 0$ which are significantly better than the result obtained for the same IRP instance without using the knowledge import at the significance level $\alpha = 0.05$. The sign '(=)' is used to mark those results for which the Wilcoxon test produced a p-value larger than 0.05. The sign '(−)' is used to mark those results obtained for $N_{\text{imp}} > 0$ which are significantly worse than the result obtained for the same IRP instance without using the knowledge import at the significance level $\alpha = 0.05$. The number of the outcomes of the statistical test is presented at the bottom of each table and is summarized in Table 2.

Table 3 Results of the Friedman statistical test with groups ("treatments") representing different values of N_{imp}

Dataset	N_{att}	Test statistic	p value
Archetti 2007 LC	10	3764.8	$< 2.2 \cdot 10^{-16}$
	$\mathcal{I}.n$	4515.2	$< 2.2 \cdot 10^{-16}$
Archetti 2007 HC	10	2985.9	$< 2.2 \cdot 10^{-16}$
	$\mathcal{I}.n$	3417.3	$< 2.2 \cdot 10^{-16}$
Hard	10	427.7	$< 2.2 \cdot 10^{-16}$
	$\mathcal{I}.n$	578.1	$< 2.2 \cdot 10^{-16}$

From the obtained results the following conclusions can be drawn:

- As shown in Table 2, the knowledge import approach managed to improve the results in about 50% of tested cases (IRP instances and values of N_{imp} and N_{att}).
- The knowledge import approach is not likely to deteriorate the optimization results. From 720 tested cases only in 4 cases a deterioration of the results was observed (Table 2).
- The *Hard* dataset contains instances for which it is indeed hard to improve the optimization results. Nevertheless, the knowledge import never deteriorated the results for this dataset, and was able to improve them in 16% of cases for $N_{att} = 10$ and in about 22% of cases for $N_{\text{att}} = \mathcal{I}.n$ (Table 2).
- For the Archetti 2007 instances, increasing N_{imp} does not seem to influence the optimization results (Tables 4, 5, 6, 7. For these instances increasing N_{att} makes a small change in the results: the number of better results increased by two between Tables 4 and 5, and between Tables 6 and 7.
- On the contrary, the number of better results obtained for the *Hard* instances increases with increasing N_{imp} and also increases for $N_{\text{att}} = \mathcal{I}.n$ compared to $N_{\text{att}} = 10$ (Tables 8 and 9).

Table 4 Results for the *Archetti 2007 LC* dataset and $N_{\text{att}} = 10$

Instance name	N_{imp} 0	1	2	5	10
abs1n5	3348.24	3348.24 (=)	3348.24 (=)	3348.24 (=)	3348.24 (=)
abs1n10	4894.11	4894.11 (=)	4894.11 (=)	4894.11 (=)	4894.11 (=)
abs1n15	6367.52	6313.71 (+)	6283.83 (+)	6219.71 (+)	6163.71 (+)
abs1n20	7108.61	7084.74 (+)	7075.75 (+)	7028.07 (+)	7028.07 (+)
abs1n25	8311.55	8265.80 (+)	8252.51 (+)	8206.39 (+)	8203.31 (+)
abs1n30	9934.90	9903.97 (+)	9881.97 (+)	9819.76 (+)	9753.40 (+)
abs2n5	2722.60	2722.60 (=)	2722.60 (=)	2722.60 (=)	2722.60 (=)
abs2n10	5773.87	5442.87 (+)	5442.87 (+)	5442.87 (+)	5442.87 (+)
abs2n15	6205.53	6184.53 (+)	6162.16 (+)	6161.16 (+)	6162.16 (+)
abs2n20	6789.76	6762.88 (=)	6753.08 (=)	6751.59 (=)	6719.26 (+)
abs2n25	8120.26	8033.94 (+)	8026.94 (+)	8026.94 (+)	8026.94 (+)
abs2n30	8873.98	8706.76 (+)	8584.98 (+)	8548.29 (+)	8548.29 (+)
abs3n5	4810.00	4810.00 (=)	4810.00 (=)	4810.00 (=)	4810.00 (=)
abs3n10	4877.53	4765.42 (+)	4654.25 (+)	4652.53 (+)	4652.53 (+)
abs3n15	6729.53	6686.53 (+)	6719.11 (=)	6698.03 (+)	6686.53 (+)
abs3n20	8025.03	7967.72 (+)	7893.26 (+)	7828.93 (+)	7802.04 (+)
abs3n25	9091.65	8960.13 (+)	8911.98 (+)	8720.13 (+)	8560.61 (+)
abs3n30	9339.63	9245.37 (+)	9183.83 (+)	9035.33 (+)	8843.91 (+)
abs4n5	3318.18	3318.18 (=)	3318.18 (=)	3318.18 (=)	3318.18 (=)
abs4n10	5487.71	5242.16 (+)	5242.16 (+)	5242.16 (+)	5242.16 (+)
abs4n15	6251.72	6251.72 (=)	6251.72 (=)	6251.72 (=)	6251.72 (=)
abs4n20	8313.22	8260.00 (+)	8241.05 (+)	8162.68 (+)	8162.62 (+)
abs4n25	8378.20	8279.20 (+)	8255.51 (+)	8249.36 (+)	8249.21 (+)
abs4n30	8777.00	8742.42 (+)	8652.58 (+)	8652.58 (+)	8652.58 (+)
abs5n5	2421.01	2421.01 (=)	2421.01 (=)	2419.67 (=)	2419.67 (+)
abs5n10	5116.79	5116.79 (=)	5126.73 (=)	5126.73 (=)	5126.73 (=)
abs5n15	5654.30	5654.30 (=)	5654.30 (=)	5654.30 (=)	5654.30 (=)
abs5n20	7958.80	7863.09 (+)	7846.09 (+)	7846.09 (+)	7846.09 (+)
abs5n25	8426.00	8368.02 (+)	8283.37 (+)	8285.07 (+)	8202.14 (+)
abs5n30	8492.44	8423.32 (+)	8384.74 (+)	8298.24 (+)	8282.06 (+)
Number of worse results (−):		0	0	0	0
Number of equal results (=):		10	11	10	8
Number of better results (+):		20	19	20	22

4.4 Comparison to other optimization methods

Experiments described in this section were aimed at comparing the evolutionary algorithm with knowledge import proposed in this paper to other methods used in the literature to solve the IRP. The methods studied in this section were as follows:

Ant Colony Optimization (ACO) proposed in the paper by Tatsis et al. (2013) for a multi-vehicle IRP, which can easily be adapted to the single-vehicle problem studied in this paper. In the aforementioned paper, a variable p_{it} denotes the position of retailer $i \in \mathcal{M}$ in the route used on day $t \in \{1, \ldots, H\}$. The ACO algorithm holds pheromone values $\tau_{\text{it}}(l)$ for $l \in 0, 1, \ldots, n$ which are used for calculating the probability that p_{it} is set to the value l:

$$P(p_{\text{it}} = l) = \frac{\tau_{it}(l)}{\sum_{k=0}^{n} \tau_{it}(k)} \ . \tag{8}$$

These probabilities are used to either exclude the retailer from the route (if $p_{\text{it}} = 0$), or to choose the position $l = p_{\text{it}}$ for retailer i in the route for day t (if $p_{\text{it}} \neq 0$). For the details of the ACO method readers are referred to the paper by Tatsis et al. (2013). Because preliminary experiments shown that the ACO method produced mostly infeasible solutions for the problem instances studied in this paper, a repair method was added consisting of the following two steps:

Table 5 Results for the *Archetti 2007 LC* dataset and $N_{\text{att}} = \mathcal{I}.n$

Instance name	N_{imp}				
	0	1	2	5	10
abs1n5	3348.24	3348.24 (=)	3348.24 (=)	3348.24 (=)	3348.24 (=)
abs1n10	4894.11	4894.11 (=)	4894.11 (=)	4894.11 (=)	4894.11 (=)
abs1n15	6367.52	6313.71 (+)	6238.03 (+)	6163.71 (+)	6163.71 (+)
abs1n20	7108.61	7084.74 (+)	7060.74 (+)	7000.46 (+)	6999.88 (+)
abs1n25	8311.55	8242.28 (+)	8203.31 (+)	8200.93 (+)	8200.93 (+)
abs1n30	9934.90	9874.45 (+)	9848.37 (+)	9762.36 (+)	9689.46 (+)
abs2n5	2722.60	2722.60 (=)	2722.60 (=)	2722.60 (=)	2722.60 (=)
abs2n10	5773.87	5442.87 (+)	5442.87 (+)	5442.87 (+)	5442.87 (+)
abs2n15	6205.53	6182.17 (+)	6162.16 (+)	6160.56 (+)	6162.16 (+)
abs2n20	6789.76	6760.05 (=)	6708.78 (+)	6708.78 (+)	6716.33 (+)
abs2n25	8120.26	8033.94 (+)	8026.94 (+)	8026.94 (+)	8026.94 (+)
abs2n30	8873.98	8570.98 (+)	8496.98 (+)	8496.98 (+)	8496.98 (+)
abs3n5	4810.00	4810.00 (=)	4810.00 (=)	4810.00 (=)	4810.00 (=)
abs3n10	4877.53	4765.42 (+)	4652.53 (+)	4652.53 (+)	4652.53 (+)
abs3n15	6729.53	6686.53 (+)	6696.55 (+)	6628.68 (+)	6657.06 (+)
abs3n20	8025.03	7951.60 (+)	7875.97 (+)	7820.10 (+)	7792.03 (+)
abs3n25	9091.65	8924.13 (+)	8855.40 (+)	8532.13 (+)	8532.13 (+)
abs3n30	9339.63	9174.00 (+)	9089.63 (+)	8808.70 (+)	8771.48 (+)
abs4n5	3318.18	3318.18 (=)	3318.18 (=)	3318.18 (=)	3318.18 (=)
abs4n10	5487.71	5242.16 (+)	5242.16 (+)	5242.16 (+)	5242.16 (+)
abs4n15	6251.72	6251.72 (=)	6251.72 (=)	6251.72 (=)	6251.72 (=)
abs4n20	8313.22	8256.00 (+)	8195.68 (+)	8161.91 (+)	8161.91 (+)
abs4n25	8378.20	8279.20 (+)	8077.81 (+)	8077.81 (+)	8077.81 (+)
abs4n30	8777.00	8652.58 (+)	8652.58 (+)	8652.58 (+)	8652.58 (+)
abs5n5	2421.01	2421.01 (=)	2421.01 (=)	2421.01 (=)	2421.01 (=)
abs5n10	5116.79	5126.73 (=)	5111.79 (=)	5111.79 (=)	5126.73 (=)
abs5n15	5654.30	5654.30 (=)	5654.30 (=)	5654.30 (=)	5654.30 (=)
abs5n20	7958.80	7846.09 (+)	7846.09 (+)	7846.09 (+)	7846.09 (+)
abs5n25	8426.00	8353.37 (+)	8223.94 (+)	8213.99 (+)	8129.67 (+)
abs5n30	8492.44	8380.58 (+)	8338.82 (+)	8280.59 (+)	8282.06 (+)
Number of worse results (−):		0	0	0	0
Number of equal results (=):		10	9	9	9
Number of better results (+):		20	21	21	21

1. For $t \in \{1, \ldots, H\}$, while the vehicle is overloaded (that is, the capacity C is exceeded), select randomly one retailer from the route for day t and remove it.
2. For $t \in \{1, \ldots, H\}$, check all retailers in a random order without repetitions and if there is a stockout at retailer i replan the schedule for this retailer using the Supply at the Latest Date (SLD) heuristic (Lipinski and Michalak 2018, 2019; Michalak 2021a).

Mathematical programming model (CPLEX) proposed in the paper by Archetti et al. (2007) in which the IRP instances *Archetti 2007 LC* and *Archetti 2007 HC* were provided. This model was used for solving the IRP using the CPLEX solver.

Population-Based Simulated Annealing (PBSA) used for solving the IRP, among others, in the paper by Shaabani and Kamalabadi (2016). In order to ensure the feasibility of solutions in the population, the population was initialized in the same way as for the evolutionary algorithm—by generating the base solution using a recursive heuristic described in Sect. 3.2 and obtaining mutated copies using the *Feasible Date Change Mutation* operator. New solutions in subsequent generations were generated using the *Feasible Date Change Mutation* and *Feasible Add Or Remove Retailer Mutation* operators.

Population-Based Tabu Search (PBTS). The Tabu Search algorithm was used for solving the IRP, among others, in

Table 6 Results for the *Archetti 2007 HC* dataset and $N_{\text{att}} = 10$

Instance name	N_{imp}				
	0	1	2	5	10
abs1n5	5955.82	5955.82 (=)	5955.82 (=)	5955.82 (=)	5955.82 (=)
abs1n10	9251.54	9233.45 (+)	9233.45 (+)	9233.45 (+)	9233.45 (+)
abs1n15	12985.26	12826.62 (+)	12760.24 (+)	12270.32 (+)	12250.00 (+)
abs1n20	15275.38	15272.99 (+)	15267.96 (+)	15268.03 (+)	15283.17 (−)
abs1n25	16752.00	16722.64 (+)	16697.47 (+)	16657.74 (+)	16650.11 (+)
abs1n30	24776.02	24742.80 (+)	24746.22 (+)	24696.44 (+)	24632.83 (+)
abs2n5	5045.91	5047.79 (=)	5047.79 (=)	5047.79 (=)	5045.91 (=)
abs2n10	9101.58	9101.58 (=)	9101.58 (=)	9101.58 (=)	9101.58 (=)
abs2n15	12635.03	12594.77 (+)	12554.36 (+)	12544.77 (+)	12436.90 (+)
abs2n20	15344.19	15308.50 (=)	15307.01 (=)	15269.50 (+)	15273.02 (+)
abs2n25	17424.68	17361.24 (+)	17354.24 (+)	17350.68 (+)	17354.24 (+)
abs2n30	21188.30	21026.87 (+)	20966.68 (+)	20906.34 (+)	20881.82 (+)
abs3n5	6990.28	6990.28 (=)	6990.28 (=)	6990.28 (=)	6990.28 (=)
abs3n10	8734.81	8734.81 (=)	8734.81 (=)	8734.81 (=)	8642.19 (+)
abs3n15	14175.77	14147.70 (+)	14144.59 (+)	14157.58 (+)	14157.58 (=)
abs3n20	15563.87	15501.40 (+)	15471.50 (+)	15396.45 (+)	15372.18 (+)
abs3n25	19346.34	19312.11 (=)	19259.19 (+)	19129.46 (+)	19061.79 (+)
abs3n30	24430.80	24367.87 (+)	24316.85 (+)	24142.98 (+)	23953.66 (+)
abs4n5	5226.16	5226.16 (=)	5226.16 (=)	5226.16 (=)	5226.16 (=)
abs4n10	9161.98	9138.40 (+)	8930.71 (+)	8930.71 (+)	8930.71 (+)
abs4n15	11374.17	11329.00 (=)	11336.25 (=)	11329.00 (=)	11329.00 (=)
abs4n20	15390.66	15359.27 (+)	15325.87 (+)	15269.46 (+)	15262.19 (+)
abs4n25	17133.41	17074.17 (+)	17057.13 (+)	17054.99 (+)	17028.49 (+)
abs4n30	18837.14	18791.66 (+)	18734.95 (+)	18733.09 (+)	18728.33 (+)
abs5n5	4593.05	4581.66 (=)	4581.66 (=)	4581.66 (=)	4581.66 (=)
abs5n10	10066.76	10066.76 (=)	10066.76 (=)	10074.49 (=)	10066.76 (=)
abs5n15	10721.92	10721.92 (=)	10721.92 (=)	10721.92 (=)	10721.92 (=)
abs5n20	16617.78	16548.52 (+)	16531.25 (+)	16520.25 (+)	16520.25 (+)
abs5n25	19914.88	19920.35 (−)	19903.24 (+)	19780.86 (+)	19776.60 (+)
abs5n30	20071.81	20029.43 (+)	19992.20 (+)	19920.43 (+)	19882.79 (+)
Number of worse results (−):		1	0	0	1
Number of equal results (=):		12	11	10	10
Number of better results (+):		17	19	20	19

papers by Alinaghian et al. (2021) and by Maghfiroh and Redi (2022). Similarly as for PBSA, the population was initialized by generating the base solution using a recursive heuristic described in Sect. 3.2 and obtaining mutated copies using the *Feasible Date Change Mutation* operator. In subsequent generations, new solutions were generated using the *Feasible Date Change Mutation* and *Feasible Add Or Remove Retailer Mutation* operators.

Particle Swarm Optimization (PSO). The PSO method was used for solving the IRP, among others, in papers by Mousavi et al. (2014) and by Wang et al. (2019). Solutions in this PSO algorithm were encoded as real vectors of length $H*n$ with n elements for each day $t \in \{1, \ldots, H\}$. The route for day t was obtained from a solution $x \in \mathbb{R}^{H*n}$ by adding retailers to the route in the descending order of the values of the elements $x[(t-1)*n+1], \ldots, x[(t-1)*n+n]$ until no more retailers could be added because of the vehicle capacity constraint. The same repair procedure as for ACO was used, because without it the PSO method generated mostly infeasible solutions.

Parameter tuning The ACO, PBSA, PBTS and PSO methods require setting some parameters, which were tuned using the same approach as described in Sect. 4.1. Because the tests involved algorithms based on different principles, the running time limit of $\max_t = 3600$ s was used as the stopping

Table 7 Results for the *Archetti 2007 HC* dataset and $N_{\text{att}} = \mathcal{I}.n$

Instance name	N_{imp}				
	0	1	2	5	10
abs1n5	5955.82	5955.82 (=)	5955.82 (=)	5955.82 (=)	5955.82 (=)
abs1n10	9251.54	9233.45 (+)	9233.45 (+)	9233.45 (+)	9233.45 (+)
abs1n15	12985.26	12812.78 (+)	12554.10 (+)	12249.68 (+)	12230.48 (+)
abs1n20	15275.38	15272.99 (+)	15273.37 (+)	15283.03 (−)	15283.17 (−)
abs1n25	16752.00	16678.79 (+)	16660.18 (+)	16651.01 (+)	16651.90 (+)
abs1n30	24776.02	24743.35 (+)	24712.96 (+)	24635.13 (+)	24525.63 (+)
abs2n5	5045.91	5045.91 (=)	5045.91 (=)	5047.79 (=)	5047.79 (=)
abs2n10	9101.58	9101.58 (=)	9101.58 (=)	9101.58 (=)	9101.58 (=)
abs2n15	12635.03	12564.51 (+)	12544.77 (+)	12537.73 (+)	12335.08 (+)
abs2n20	15344.19	15281.38 (+)	15284.53 (+)	15265.35 (+)	15273.33 (+)
abs2n25	17424.68	17354.24 (+)	17350.68 (+)	17350.68 (+)	17354.24 (+)
abs2n30	21188.30	20900.09 (+)	20831.50 (+)	20831.50 (+)	20831.50 (+)
abs3n5	6990.28	6990.28 (=)	6990.28 (=)	6990.28 (=)	6990.28 (=)
abs3n10	8734.81	8734.81 (=)	8734.81 (=)	8734.81 (=)	8734.81 (=)
abs3n15	14175.77	14158.81 (+)	14197.18 (=)	14142.59 (+)	14157.58 (=)
abs3n20	15563.87	15504.35 (+)	15433.46 (+)	15375.45 (+)	15367.24 (+)
abs3n25	19346.34	19268.04 (+)	19105.38 (+)	19018.93 (+)	19043.17 (+)
abs3n30	24430.80	24319.57 (+)	24229.18 (+)	23935.29 (+)	23900.10 (+)
abs4n5	5226.16	5226.16 (=)	5226.16 (=)	5226.16 (=)	5226.16 (=)
abs4n10	9161.98	9138.40 (+)	8930.71 (+)	8930.71 (+)	8930.71 (+)
abs4n15	11374.17	11329.21 (=)	11333.06 (=)	11333.06 (=)	11329.00 (+)
abs4n20	15390.66	15357.67 (+)	15298.29 (+)	15261.42 (+)	15257.19 (+)
abs4n25	17133.41	17046.66 (+)	17023.80 (+)	16890.75 (+)	16890.75 (+)
abs4n30	18837.14	18733.02 (+)	18733.35 (+)	18733.02 (+)	18734.95 (+)
abs5n5	4593.05	4581.66 (=)	4581.66 (=)	4593.05 (=)	4581.66 (=)
abs5n10	10066.76	10066.76 (=)	10066.76 (=)	10066.76 (=)	10072.49 (=)
abs5n15	10721.92	10721.92 (=)	10721.92 (=)	10721.92 (=)	10721.92 (=)
abs5n20	16617.78	16520.25 (+)	16520.25 (+)	16520.25 (+)	16520.25 (+)
abs5n25	19914.88	19805.34 (+)	19906.60 (+)	19754.56 (+)	19668.79 (+)
abs5n30	20071.81	19979.09 (+)	19935.51 (+)	19877.49 (+)	19883.25 (+)
Number of worse results (−):		0	0	1	1
Number of equal results (=):		10	11	10	10
Number of better results (+):		20	19	19	19

criterion instead of the number of solution evaluations. The candidate values for these parameters were as follows:

- For all algorithms: the population size $N_{\text{pop}} \in \{50, 100, 200, 500\}$
- For ACO: the pheromone evaporation rate $R_{\text{ev}} \in \{0.001, 0.002, 0.005, 0.010\}$, which controls how fast the algorithm "forgets" the previous state, the pheromone increment $\Delta\tau \in \{0.01, 0.02, 0.05, 0.10, 0.20\}$, which controls how fast the algorithm incorporates new information from good solutions, and the pheromone restart frequency $\nu_{\text{re}} \in \{0.02, 0.05, 0.10, 0.20, 0.50\}$, which controls how often the pheromones are reset to initial values (with the value $\nu_{\text{re}} = 0.1$, for example, meaning that the reset occurs every 10% of the algorithm running time).
- For PBSA: the selection scheme used when selecting parents for generating new solutions. In the *Same* scheme, each solution $x_i, i = 1, \ldots, N_{\text{pop}}$ in the existing population is used as a parent for generating a new solution using mutation and if the new solution is better than x_i, then x_i is replaced with the new one. In the *Random* scheme when trying to generate a new solution to replace x_i, the parent solution is selected randomly from the whole population, which helps disseminating information about good solutions in the population.

Table 8 Results for the *Hard* dataset and $N_{\text{att}} = 10$

Instance name	N_{imp}				
	0	1	2	5	10
hard1n10	5982.16	5982.16 (=)	5982.16 (=)	5982.16 (=)	5982.16 (=)
hard1n15	5835.34	5835.34 (=)	5835.34 (=)	5835.34 (=)	5835.34 (=)
hard1n20	8030.17	8030.17 (=)	8030.17 (=)	8030.17 (=)	8030.17 (=)
hard1n25	9228.18	9228.18 (=)	9228.31 (=)	9223.00 (=)	9223.00 (=)
hard1n30	5939.37	5942.53 (=)	5939.11 (=)	5931.77 (=)	5938.92 (=)
hard1n35	5918.51	5918.18 (+)	5918.06 (+)	5917.44 (+)	5917.19 (+)
hard2n10	5912.57	5913.25 (=)	5911.89 (=)	5911.89 (=)	5911.89 (=)
hard2n15	7682.56	7682.56 (=)	7682.56 (=)	7682.56 (=)	7682.56 (=)
hard2n20	6950.62	6950.62 (=)	6950.62 (=)	6950.62 (=)	6950.62 (=)
hard2n25	5241.90	5241.90 (=)	5241.90 (=)	5241.90 (=)	5241.90 (=)
hard2n30	9574.90	9573.54 (+)	9573.54 (+)	9573.54 (+)	9573.54 (+)
hard2n35	9488.66	9471.81 (=)	9486.15 (=)	9476.33 (=)	9486.15 (=)
hard3n10	7164.72	7164.72 (=)	7164.72 (=)	7164.72 (=)	7164.72 (=)
hard3n15	5958.00	5958.00 (=)	5958.00 (=)	5958.00 (=)	5958.00 (=)
hard3n20	9535.86	9535.86 (=)	9535.86 (=)	9535.86 (=)	9535.86 (=)
hard3n25	6276.03	6282.64 (=)	6274.61 (=)	6273.89 (+)	6273.89 (=)
hard3n30	8965.83	8965.83 (=)	8965.83 (=)	8948.87 (+)	8948.87 (+)
hard3n35	5673.66	5663.46 (+)	5658.52 (+)	5650.46 (+)	5650.46 (+)
hard4n10	7091.61	7091.61 (=)	7091.61 (=)	7091.61 (=)	7091.61 (=)
hard4n15	6824.70	6824.70 (=)	6824.70 (=)	6824.70 (=)	6824.70 (=)
hard4n20	8381.88	8381.88 (=)	8381.88 (=)	8381.88 (=)	8381.88 (=)
hard4n25	7657.89	7657.89 (=)	7657.89 (=)	7657.89 (=)	7657.89 (=)
hard4n30	11419.86	11419.97 (=)	11419.86 (=)	11419.97 (=)	11419.86 (=)
hard4n35	5053.19	5053.03 (=)	5051.00 (+)	5049.51 (+)	5045.12 (+)
hard5n10	5131.55	5131.55 (=)	5131.55 (=)	5131.55 (=)	5131.55 (=)
hard5n15	6951.92	6951.92 (=)	6951.92 (=)	6951.92 (=)	6951.92 (=)
hard5n20	4935.14	4935.14 (=)	4935.14 (=)	4935.14 (=)	4935.14 (=)
hard5n25	10734.47	10734.47 (=)	10734.47 (=)	10734.47 (=)	10734.47 (=)
hard5n30	5466.25	5466.25 (=)	5466.25 (=)	5465.78 (+)	5465.78 (+)
hard5n35	5715.46	5714.43 (=)	5715.62 (=)	5714.28 (=)	5715.20 (=)
Number of worse results (−):		0	0	0	0
Number of equal results (=):		27	26	23	24
Number of better results (+):		3	4	7	6

– For PBTS: the selection scheme used when selecting parents for generating new solutions, which can be *Random* or *Same* similarly as for PBSA, the number of candidate solutions generated in each algorithm iteration $N_{\text{new}} \in \{10, 20, 50, 100\}$, and the tabu list length $N_{\text{tabu}} \in \{100, 200, 500, 1000, 2000, 5000, 10000\}$.
– For PSO: the inertia weight $w \in \{0.2, 0.4, 0.6, 0.8, 1.0\}$, the cognitive coefficient $\phi_p \in \{1.0, 1.5, 2.0, 2.5, 3.0\}$, and the social coefficient $\phi_g \in \{1.0, 1.5, 2.0, 2.5, 3.0\}$.

The best parameter settings for each method were determined to be:

– For ACO: the population size $N_{\text{pop}} = 500$, the pheromone evaporation rate $R_{\text{ev}} = 0.002$, the pheromone increment $\Delta\tau = 0.10$, and the pheromone restart frequency $\nu_{\text{re}} = 0.05$.
– For PBSA: the population size $N_{\text{pop}} = 200$, and the *Random* parent selection scheme
– For PBTS: the population size $N_{\text{pop}} = 200$, the *Random* parent selection scheme, the number of candidate solutions generated in each algorithm iteration $N_{\text{new}} = 20$, and the tabu list length $N_{\text{tabu}} = 200$.
– For PSO: the population size $N_{\text{pop}} = 50$, the inertia weight $w = 0.8$, the cognitive coefficient $\phi_p = 1.0$, and the social coefficient $\phi_g = 1.5$.

Table 9 Results for the *Hard* dataset and $N_{att} = \mathcal{I}.n$

Instance name	N_{imp}				
	0	1	2	5	10
hard1n10	5982.16	5982.16 (=)	5982.16 (=)	5982.16 (=)	5982.16 (=)
hard1n15	5835.34	5835.34 (=)	5835.34 (=)	5835.34 (=)	5835.34 (=)
hard1n20	8030.17	8030.17 (=)	8030.17 (=)	8030.17 (=)	8030.17 (=)
hard1n25	9228.18	9229.00 (=)	9228.18 (=)	9223.00 (=)	9223.00 (=)
hard1n30	5939.37	5943.30 (=)	5932.73 (=)	5932.96 (=)	5939.24 (=)
hard1n35	5918.51	5917.63 (+)	5917.33 (+)	5917.19 (+)	5917.19 (+)
hard2n10	5912.57	5911.89 (=)	5911.89 (=)	5911.89 (=)	5911.89 (=)
hard2n15	7682.56	7682.56 (=)	7682.56 (=)	7682.56 (=)	7682.56 (=)
hard2n20	6950.62	6950.62 (=)	6950.62 (=)	6950.62 (=)	6950.62 (=)
hard2n25	5241.90	5241.90 (=)	5241.90 (=)	5241.90 (=)	5241.90 (=)
hard2n30	9574.90	9573.54 (+)	9573.54 (+)	9573.54 (+)	9568.57 (+)
hard2n35	9488.66	9486.15 (=)	9478.74 (=)	9456.06 (=)	9454.88 (+)
hard3n10	7164.72	7164.72 (=)	7164.72 (=)	7164.72 (=)	7164.72 (=)
hard3n15	5958.00	5958.00 (=)	5958 (=)	5958.00 (=)	5958.00 (=)
hard3n20	9535.86	9535.86 (=)	9535.86 (=)	9535.86 (=)	9535.86 (=)
hard3n25	6276.03	6274.61 (=)	6274.61 (=)	6273.89 (=)	6273.89 (+)
hard3n30	8965.83	8948.87 (+)	8948.87 (+)	8948.87 (+)	8948.87 (+)
hard3n35	5673.66	5650.46 (+)	5650.46 (+)	5650.46 (+)	5650.46 (+)
hard4n10	7091.61	7091.61 (=)	7091.61 (=)	7091.61 (=)	7091.61 (=)
hard4n15	6824.70	6824.70 (=)	6824.70 (=)	6824.70 (=)	6824.70 (=)
hard4n20	8381.88	8381.88 (=)	8381.88 (=)	8381.88 (=)	8381.88 (=)
hard4n25	7657.89	7657.89 (=)	7657.89 (=)	7647.16 (=)	7657.89 (=)
hard4n30	11419.86	11419.86 (=)	11419.86 (=)	11419.97 (=)	11419.97 (=)
hard4n35	5053.19	5049.43 (+)	5050.92 (=)	5045.12 (+)	5044.42 (+)
hard5n10	5131.55	5131.55 (=)	5131.55 (=)	5131.55 (=)	5131.55 (=)
hard5n15	6951.92	6951.92 (=)	6951.92 (=)	6951.92 (=)	6951.92 (=)
hard5n20	4935.14	4935.14 (=)	4935.14 (=)	4935.14 (=)	4935.14 (=)
hard5n25	10734.47	10734.47 (=)	10734.47 (=)	10734.47 (=)	10734.47 (=)
hard5n30	5466.25	5466.25 (=)	5465.78 (+)	5465.78 (+)	5465.78 (+)
hard5n35	5715.46	5714.28 (=)	5714.28 (+)	5714.28 (+)	5714.28 (+)
Number of worse results (−):		0	0	0	0
Number of equal results (=):		25	24	23	21
Number of better results (+):		5	6	7	9

The ACO, CPLEX, PBSA, PBTS and PSO methods were compared to the evolutionary algorithm with knowledge import proposed in this paper with the parameters obtained in Sect. 4.1: the Elitism reduction mode, N_{pop} = 200, P_{cross} = 0.8, P_{mut} = 0.1. The number of knowledge imports was set to $N_{imp} = 1$ and the number of attempts to improve the solution in each knowledge import set to $N_{att} = 10$.

Method comparison Tested methods, with the parameter settings listed above, were compared by running each method 30 times for each IRP instance used in this paper: 30 *Archetti 2007 LC* instances, 30 *Archetti 2007 HC* instances, and 30 *Hard* instances. In order to test if the choice of the optimization method affected the quality of the solutions, the Friedman test was applied to the results obtained for each dataset and each optimization method. The null hypothesis of this test is that the medians of the optimization results are equal across all groups (optimization methods). The results of the Friedman test are presented in Table 10. Based on low p-values obtained in this test it can be concluded that there are statistically significant differences of solution quality among different optimization methods. Detailed results are presented in Tables 11, 12, 13 in which median values obtained in 30 runs of the optimization methods are shown.

In the column *Knowledge-based EA*, the results attained by the method proposed in this paper are presented. The following columns contain the results attained by the com-

Table 10 Results of the Friedman statistical test with groups ("treatments") representing different optimization methods

Dataset	Test statistic	p value
Archetti 2007 LC	2421.6	$< 2.2 \cdot 10^{-16}$
Archetti 2007 HC	2244.9	$< 2.2 \cdot 10^{-16}$
Hard	2028.3	$< 2.2 \cdot 10^{-16}$

parison methods: ACO, CPLEX, PBSA, PBTS and PSO described at the beginning of this section. The sign '(+)' is used to mark those results obtained by comparison methods which are significantly better than the result obtained for the same IRP instance using the knowledge-based EA at the significance level $\alpha = 0.05$. The sign '(=)' is used to mark those results for which the Wilcoxon test produced a p-value larger than 0.05. The sign '(−)' is used to mark those results obtained by comparison methods which are significantly worse than the result obtained for the same IRP instance using the knowledge-based EA at the significance level $\alpha = 0.05$. The footer of each table shows the number of times each comparison method attained a worse, equal, or better result in comparison to the knowledge-based EA. From the obtained results the following conclusions can be drawn:

Table 11 Method comparison for the *Archetti 2007 LC* dataset

Instance name	Algorithm					
	Knowledge-based EA	ACO	CPLEX	PBSA	PBTS	PSO
abs1n5	3348.24	3361.39 (=)	3335.24 (=)	3350.67 (=)	3350.67 (−)	3350.67 (=)
abs1n10	4894.11	4894.11 (=)	4499.25 (+)	4894.11 (=)	4894.11 (=)	4888.85 (=)
abs1n15	6259.48	6388.71 (−)	5462.68 (+)	6388.71 (−)	6388.71 (−)	5976.74 (+)
abs1n20	7085.58	7109.45 (−)	6704.09 (+)	7109.45 (−)	7109.45 (−)	7327.77 (−)
abs1n25	8309.05	8842.79 (−)	7095.86 (+)	8711.05 (−)	8711.05 (−)	8469.56 (−)
abs1n30	9873.55	10499.80 (−)	10166.01 (=)	10367.54 (−)	10367.54 (−)	9707.17 (+)
abs2n5	2722.60	2793.47 (−)	2722.33 (+)	2739.83 (=)	2722.33 (=)	2740.33 (−)
abs2n10	5442.87	5773.87 (−)	5236.98 (+)	5773.87 (−)	5773.87 (−)	5937.93 (−)
abs2n15	6188.54	6689.75 (−)	5494.74 (+)	6691.65 (−)	6691.65 (−)	6195.51 (=)
abs2n20	6726.21	7511.99 (−)	6094.14 (+)	7475.22 (−)	7486.26 (−)	6798.37 (=)
abs2n25	8061.07	8121.94 (−)	8477.28 (−)	8121.94 (−)	8121.94 (−)	8409.53 (−)
abs2n30	8715.98	8875.98 (−)	8395.51 (+)	8875.98 (−)	8875.98 (−)	8989.88 (−)
abs3n5	4810.00	4810.00 (=)	4776.00 (+)	4810.00 (=)	4810.00 (=)	4843.00 (−)
abs3n10	4765.42	5173.07 (−)	4652.53 (=)	5173.07 (−)	5163.43 (−)	4877.53 (=)
abs3n15	6722.68	7361.69 (−)	6060.38 (+)	7299.74 (−)	7239.80 (−)	6865.59 (−)
abs3n20	7983.88	8554.13 (−)	6950.20 (+)	8577.53 (−)	8563.53 (−)	8131.10 (−)
abs3n25	8962.83	9723.92 (−)	8202.80 (+)	9742.52 (−)	9742.52 (−)	9115.80 (=)
abs3n30	9313.26	10046.80 (−)	8543.19 (+)	9823.01 (−)	9823.01 (−)	11084.46 (−)
abs4n5	3318.18	3389.66 (=)	3246.66 (=)	3318.18 (=)	3318.18 (=)	3389.66 (=)
abs4n10	5242.16	5487.71 (−)	5104.91 (+)	5487.71 (−)	5487.71 (−)	5440.46 (−)
abs4n15	6251.72	6252.60 (=)	5504.65 (+)	6252.60 (=)	6252.60 (=)	6448.69 (−)
abs4n20	8280.14	8364.53 (−)	7590.89 (+)	8364.53 (−)	8364.53 (−)	8549.66 (−)
abs4n25	8295.39	8378.20 (−)	7509.02 (+)	8378.20 (−)	8378.20 (−)	9170.21 (−)
abs4n30	8677.03	8777.03 (=)	7735.25 (+)	8777.03 (=)	8777.03 (=)	10724.31 (−)
abs5n5	2421.01	2561.84 (−)	2419.67 (=)	2498.52 (=)	2496.18 (=)	2546.11 (−)
abs5n10	5143.56	5212.79 (=)	4670.76 (+)	5212.79 (=)	5212.79 (=)	5129.01 (=)
abs5n15	5654.30	5654.30 (=)	5309.48 (+)	5654.30 (=)	5654.30 (=)	6052.77 (−)
abs5n20	7846.09	7959.40 (−)	7512.40 (+)	7959.40 (−)	7959.40 (−)	8297.21 (−)
abs5n25	8280.87	8959.27 (−)	7568.11 (+)	8901.43 (−)	8901.43 (−)	8524.30 (−)
abs5n30	8437.24	8862.82 (−)	8160.22 (=)	8824.32 (−)	8824.32 (−)	8589.42 (−)
Number of worse results (−):		22	1	20	21	20
Number of equal results (=):		8	6	10	9	8
Number of better results (+):		0	23	0	0	2

Table 12 Method comparison for the *Archetti 2007 HC* dataset

Instance name	Algorithm					
	Knowledge-based EA	ACO	CPLEX	PBSA	PBTS	PSO
abs1n5	3348.24	3361.39 (=)	3335.24 (=)	3350.67 (=)	3350.67 (−)	3350.67 (=)
abs1n10	4894.11	4894.11 (=)	4499.25 (+)	4894.11 (=)	4894.11 (=)	4888.85 (=)
abs1n15	6259.48	6388.71 (−)	5462.68 (+)	6388.71 (−)	6388.71 (−)	5976.74 (+)
abs1n20	7085.58	7109.45 (−)	6704.09 (+)	7109.45 (−)	7109.45 (−)	7327.77 (−)
abs1n25	8309.05	8842.79 (−)	7095.86 (+)	8711.05 (−)	8711.05 (−)	8469.56 (−)
abs1n30	9873.55	10499.80 (−)	10166.01 (=)	10367.54 (−)	10367.54 (−)	9707.17 (+)
abs2n5	2722.60	2793.47 (−)	2722.33 (+)	2739.83 (=)	2722.33 (=)	2740.33 (−)
abs2n10	5442.87	5773.87 (−)	5236.98 (+)	5773.87 (−)	5773.87 (−)	5937.93 (−)
abs2n15	6188.54	6689.75 (−)	5494.74 (+)	6691.65 (−)	6691.65 (−)	6195.51 (=)
abs2n20	6726.21	7511.99 (−)	6094.14 (+)	7475.22 (−)	7486.26 (−)	6798.37 (=)
abs2n25	8061.07	8121.94 (−)	8477.28 (−)	8121.94 (−)	8121.94 (−)	8409.53 (−)
abs2n30	8715.98	8875.98 (−)	8395.51 (+)	8875.98 (−)	8875.98 (−)	8989.88 (−)
abs3n5	4810.00	4810.00 (=)	4776.00 (+)	4810.00 (=)	4810.00 (=)	4843.00 (−)
abs3n10	4765.42	5173.07 (−)	4652.53 (=)	5173.07 (−)	5163.43 (−)	4877.53 (=)
abs3n15	6722.68	7361.69 (−)	6060.38 (+)	7299.74 (−)	7239.80 (−)	6865.59 (−)
abs3n20	7983.88	8554.13 (−)	6950.20 (+)	8577.53 (−)	8563.53 (−)	8131.10 (−)
abs3n25	8962.83	9723.92 (−)	8202.80 (+)	9742.52 (−)	9742.52 (−)	9115.80 (=)
abs3n30	9313.26	10046.80 (−)	8543.19 (+)	9823.01 (−)	9823.01 (−)	11084.46 (−)
abs4n5	3318.18	3389.66 (=)	3246.66 (=)	3318.18 (=)	3318.18 (=)	3389.66 (=)
abs4n10	5242.16	5487.71 (−)	5104.91 (+)	5487.71 (−)	5487.71 (−)	5440.46 (−)
abs4n15	6251.72	6252.60 (=)	5504.65 (+)	6252.60 (=)	6252.60 (=)	6448.69 (−)
abs4n20	8280.14	8364.53 (−)	7590.89 (+)	8364.53 (−)	8364.53 (−)	8549.66 (−)
abs4n25	8295.39	8378.20 (−)	7509.02 (+)	8378.20 (−)	8378.20 (−)	9170.21 (−)
abs4n30	8677.03	8777.03 (=)	7735.25 (+)	8777.03 (=)	8777.03 (=)	10724.31 (−)
abs5n5	2421.01	2561.84 (−)	2419.67 (=)	2498.52 (=)	2496.18 (=)	2546.11 (−)
abs5n10	5143.56	5212.79 (=)	4670.76 (+)	5212.79 (=)	5212.79 (=)	5129.01 (=)
abs5n15	5654.30	5654.30 (=)	5309.48 (+)	5654.30 (=)	5654.30 (=)	6052.77 (−)
abs5n20	7846.09	7959.40 (−)	7512.40 (+)	7959.40 (−)	7959.40 (−)	8297.21 (−)
abs5n25	8280.87	8959.27 (−)	7568.11 (+)	8901.43 (−)	8901.43 (−)	8524.30 (−)
abs5n30	8437.24	8862.82 (−)	8160.22 (=)	8824.32 (−)	8824.32 (−)	8589.42 (−)
Number of worse results (−):		22	1	20	21	20
Number of equal results (=):		8	6	10	9	8
Number of better results (+):		0	23	0	0	2

– The knowledge-based EA proposed in this paper is the best performing metaheuristic algorithm among the ones tested in the experiments. It was never outperformed by ACO nor by PBSA and was outperformed in 2 of 30 tests on the *Archetti 2007 LC* dataset by PSO, in 2 of 30 tests on the *Archetti 2007 HC* dataset by PSO again, and 1 of 30 tests on the *Hard* dataset by PBTS.
– For the *Archetti 2007 LC* and *Archetti 2007 HC* datasets, all metaheuristic algorithms were outperformed by the mathematical programming model (CPLEX), which attained the best result in 23 of 30 tests (in the case of both datasets). This is not surprising, because this mathematical model was proposed in the same paper by Archetti et al. (2007) in which these IRP instances were provided, so it is undoubtedly well suited for them.
– For the *Hard* dataset the knowledge-based EA proposed in this paper is the best optimization algorithm of all the tested methods. It outperformed the mathematical programming model (CPLEX) in 24 of 30 tests with the remaining 6 resulting in a draw. The CPLEX method never attained a result better than the knowledge-based EA on the *Hard* dataset. ACO, PBSA,

Table 13 Method comparison for the *Hard* dataset

Instance name	Algorithm					
	Knowledge-based EA	ACO	CPLEX	PBSA	PBTS	PSO
hard1n10	5982.16	6001.78 (−)	5982.16 (=)	5982.16 (=)	5982.16 (=)	5990.41 (−)
hard1n15	5839.34	5960.01 (−)	5870.59 (−)	5942.59 (−)	5957.57 (−)	5866.85 (=)
hard1n20	8030.17	8287.53 (−)	8154.94 (−)	8178.24 (−)	8236.34 (−)	7995.11 (=)
hard1n25	9211.59	9759.42 (−)	9623.67 (−)	9554.60 (−)	9619.90 (−)	9239.59 (=)
hard1n30	5938.02	6247.63 (−)	6116.10 (−)	6145.01 (−)	6153.98 (−)	5980.74 (−)
hard1n35	5918.20	6062.27 (−)	6000.35 (−)	6062.27 (−)	6062.27 (−)	5956.52 (−)
hard2n10	5913.25	5925.64 (−)	5911.11 (=)	5911.11 (=)	5911.11 (=)	5911.11 (=)
hard2n15	7682.56	7725.91 (−)	7709.48 (=)	7687.84 (−)	7718.74 (−)	7683.56 (−)
hard2n20	6950.62	7309.95 (−)	7062.35 (−)	7172.59 (−)	7209.77 (−)	7029.64 (−)
hard2n25	5241.90	5370.39 (−)	5277.29 (−)	5309.48 (−)	5324.66 (−)	5241.90 (=)
hard2n30	9574.90	9930.73 (−)	9760.15 (−)	9835.55 (−)	9868.75 (−)	9571.57 (=)
hard2n35	9486.15	9849.61 (−)	9930.01 (−)	9771.97 (−)	9799.69 (−)	9534.49 (−)
hard3n10	7164.72	7176.22 (−)	7164.72 (=)	7164.72 (=)	7167.02 (=)	7184.91 (−)
hard3n15	5958.00	6011.49 (−)	6001.75 (−)	5963.41 (−)	5963.49 (−)	5988.34 (=)
hard3n20	9535.86	9757.45 (−)	9682.51 (−)	9653.92 (−)	9619.02 (−)	10860.03 (−)
hard3n25	6275.31	6451.35 (−)	6374.73 (−)	6370.38 (−)	6365.16 (−)	6301.60 (−)
hard3n30	8954.85	9383.99 (−)	9336.51 (−)	9383.99 (−)	9381.45 (−)	8965.83 (=)
hard3n35	5658.52	5928.44 (−)	5791.98 (−)	5928.44 (−)	5928.44 (−)	5686.92 (−)
hard4n10	7091.61	7102.59 (=)	7091.61 (=)	7091.61 (=)	7101.93 (=)	7102.11 (=)
hard4n15	6824.70	6865.36 (−)	6830.85 (−)	6815.92 (=)	6815.92 (+)	6830.64 (−)
hard4n20	8381.88	8446.70 (−)	8455.95 (−)	8421.95 (−)	8395.58 (−)	8730.72 (−)
hard4n25	7674.69	8245.05 (−)	7755.14 (−)	7992.47 (−)	8087.86 (−)	7710.09 (=)
hard4n30	11419.97	11581.73 (−)	11824.83 (−)	11610.95 (−)	11620.17 (−)	11420.01 (=)
hard4n35	5056.25	5698.86 (−)	5413.32 (−)	5557.66 (−)	5633.74 (−)	5062.40 (=)
hard5n10	5131.55	5156.85 (−)	5131.55 (=)	5131.55 (=)	5149.36 (=)	5131.55 (=)
hard5n15	6951.92	7068.16 (−)	7033.92 (−)	6989.35 (−)	6999.06 (−)	6978.16 (=)
hard5n20	4935.14	5162.83 (−)	5013.63 (−)	4982.84 (−)	4990.32 (−)	5006.66 (−)
hard5n25	10734.47	11039.89 (−)	11051.74 (−)	10883.11 (−)	10950.99 (−)	10739.21 (=)
hard5n30	5466.25	5693.29 (−)	5557.05 (−)	5663.32 (−)	5655.72 (−)	5486.22 (−)
hard5n35	5720.19	6343.24 (−)	5857.00 (−)	6143.85 (−)	6219.82 (−)	5826.61 (−)
Number of worse results (−):		29	24	24	24	15
Number of equal results (=):		1	6	6	5	15
Number of better results (+):		0	0	0	1	0

and PBTS performed poorly in comparison to the proposed knowledge-based EA attaining worse results in at least 24 of 30 tests. PSO attained worse results in 15 of 30 tests, while in the remaining 15 tests it attained results equal to the knowledge-based EA. However, similarly as ACO, CPLEX, and PBSA it never attained results better than the knowledge-based EA. The only method that outperformed the knowledge-based EA on the *Hard* dataset was PBTS, and it was in just 1 of 30 tests.

5 Conclusion

In this paper, evolutionary optimization using an external knowledge source was proposed for the Inventory Routing Problem (IRP). During optimization, the algorithm performs N_{imp} knowledge imports which modify schedules for individual retailers in the best solution x_{best} found so far by the evolutionary algorithm. In each import, at most N_{att} attempts to improve the solution are performed. In each attempt, one retailer is randomly selected (without repetitions) and the schedule for this retailer is imported to x_{best} from the knowledge source. From the N_{att} attempts to improve the solution

the best one x^* is selected and if it is better than x_{best} it replaces the *worst* solution in the EA population. The knowledge can be imported from either a known good solution to the optimization problem, or by interacting with the user. In the former case the schedules for the selected retailers are copied from the known good solution, and in the latter case the user is expected to modify these schedules according to his/her expertise in the problem domain.

In the experiments, in which the evolutionary algorithm using the knowledge import mechanism was compared to an algorithm not using this mechanism, it was observed that the knowledge import mechanism was not likely to deteriorate the results (a worse result was only observed in 4 cases out of 720). In about 50% of tested cases the knowledge import improved the optimization results and in the remaining cases the results were not statistically different for the algorithm with and without the knowledge import. For the *Hard* IRP instances proposed in the paper by Michalak (2021b) the knowledge import mechanism never deteriorated the results, and it was able to improve the results in 16–22% of cases (depending on the number of attempts to improve the solution in each knowledge import N_{att}). These IRP instances seem to be indeed very hard to tackle. For the IRP instances proposed in the paper by Archetti et al. (2007) the knowledge import mechanism improved the results in about 66% of cases. For the *Archetti 2007 LC* dataset this rate of improvement was observed for $N_{\text{imp}} = 1$ and $N_{\text{att}} = 10$. This means, that if the user can be expected to provide 10 good schedules for individual retailers in one interactive session during the entire algorithm's runtime the optimization results can be improved in about 66% of cases. Therefore, the workload of the user should not be too high, and the proposed approach appears to be suitable for designing user-interactive optimization methods for the IRP.

In the experiments, in which the knowledge-based EA was compared to other optimization methods (ACO, PBSA, PBTS, PSO, and a mathematical programming model solved using CPLEX), it outperformed all the metaheuristics. The CPLEX-based method performed better than the knowledge-based EA on the *Archetti 2007 LC* and *Archetti 2007 HC* datasets. On the other hand, on the *Hard* instances the knowledge-based EA performed better than all comparison methods including the mathematical programming model.

To sum up, the proposed knowledge import mechanism can be used to improve optimization results by either importing knowledge from a known good solution to the IRP instance, or by interacting with the user, who can modify the best solution found by the optimization algorithm. This aspect of the proposed method can be used to the benefit of practitioners, who have the required know-how to improve the delivery schedules or want to follow good practices established at their company. Another advantage of the proposed method is that it is able to improve the results for very hard IRP instances. In the experiments, it was found to be the best-performing optimization method for the *Hard* instances. On the other hand, it turned out not to be competitive to the mathematical programming model on easier IRP instances, but this is an observation true for all the tested metaheuristics.

A certain limitation of the presented work is that it did not address more complex variants of the IRP, but, given the large variety of inventory routing problems studied in the literature, each version would require a separate extensive study and discussion of the results in a dedicated paper. Therefore, adapting the proposed method to various types of the IRP used in practical applications was left for future work.

Funding This work was supported by the Polish National Science Centre under Grant No. 2015/19/D/HS4/02574. Calculations have been carried out using resources provided by Wrocław Centre for Networking and Supercomputing (http://wcss.pl), Grant No. 407.

Data availability The datasets generated during and/or analysed during the current study are available from the corresponding author on reasonable request.

Declarations

Conflict of interest The authors declare that they have no conflict of interest.

References

Aghezzaf EH, Raa B, Van Landeghem H (2006) Modeling inventory routing problems in supply chains of high consumption products. Eur J Oper Res 169(3):1048–1063

Agra A, Christiansen M, Wolsey L (2022) Improved models for a single vehicle continuous-time inventory routing problem with pickups and deliveries. Eur J Oper Res 297(1):164–179

Alinaghian M, Tirkolaee EB, Dezaki ZK, Hejazi SR, Ding W (2021) An augmented tabu search algorithm for the green inventory-routing problem with time windows. Swarm Evol Comput 60:100802

Alkaabneh F, Diabat A, Gao HO (2020) Benders decomposition for the inventory vehicle routing problem with perishable products and environmental costs. Comput Oper Res 113:104751

Archetti C, Ljubić I (2022) Comparison of formulations for the inventory routing problem. Eur J Oper Res 303:997

Archetti C, Bertazzi L, Laporte G, Speranza MG (2007) A branch-and-cut algorithm for a vendor-managed inventory-routing problem. Transp Sci 41(3):382–391

Bard JF, Nananukul N (2009) Heuristics for a multiperiod inventory routing problem with production decisions. Comput Ind Eng 57(3):713–723

Bertazzi L, Speranza MG (2012) Inventory routing problems: an introduction. EURO J Transp Log 1(4):307–326

Bertazzi L, Bosco A, Guerriero F, Lagana D (2013) A stochastic inventory routing problem with stock-out. Transp Res Part C Emerg Technol 27:89–107

Blumenfeld DE, Burns LD, Diltz J, Daganzo CF (1985) Analyzing trade-offs between transportation, inventory and production costs on freight networks. Transp Res Part B Methodol 19(5):361–380

Burns LD, Hall RW, Blumenfeld DE, Daganzo CF (1985) Distribution strategies that minimize transportation and inventory costs. Oper Res 33(3):469–490

Çelik M, Archetti C, Süral H (2022) Inventory routing in a warehouse: the storage replenishment routing problem. Eur J Oper Res 301(3):1117–1132

Dantzig GB, Ramser JH (1959) The truck dispatching problem. Manag Sci 6(1):80–91

De A, Kumar SK, Gunasekaran A, Tiwari MK (2017) Sustainable maritime inventory routing problem with time window constraints. Eng Appl Artif Intell 61:77–95

De M, Giri BC (2020) Modelling a closed-loop supply chain with a heterogeneous fleet under carbon emission reduction policy. Transp Res Part E Logist Transp Rev 133:101813

Dev NK, Shankar R, Choudhary A (2017) Strategic design for inventory and production planning in closed-loop hybrid systems. Int J Prod Econ 183:345–353

Diabat A, Dehghani E, Jabbarzadeh A (2017) Incorporating location and inventory decisions into a supply chain design problem with uncertain demands and lead times. J Manuf Syst 43:139–149

Hiassat A, Diabat A, Rahwan I (2017) A genetic algorithm approach for location-inventory-routing problem with perishable products. J Manuf Syst 42:93–103

Iassinovskaia G, Limbourg S, Riane F (2017) The inventory-routing problem of returnable transport items with time windows and simultaneous pickup and delivery in closed-loop supply chains. Int J Prod Econ 183:570–582

Juan AA, Grasman SE, Caceres-Cruz J, Bektaş T (2014) A simheuristic algorithm for the single-period stochastic inventory-routing problem with stock-outs. Simul Model Pract Theory 46:40–52

Juan AA, Faulin J, Grasman SE, Rabe M, Figueira G (2015) A review of simheuristics: extending metaheuristics to deal with stochastic combinatorial optimization problems. Oper Res Persp 2:62–72

Lagos F, Boland N, Savelsbergh M (2020) The continuous-time inventory-routing problem. Transp Sci 54(2):375–399

Lagos F, Boland N, Savelsbergh M (2022) Dynamic discretization discovery for solving the continuous time inventory routing problem with out-and-back routes. Comput Oper Res 141:105686

Laporte G (2009) Fifty years of vehicle routing. Transp Sci 43(4):408–416

Lipinski P, Michalak K (2018) An evolutionary algorithm with practitioner's-knowledge-based operators for the inventory routing problem. In: Liefooghe A, López-Ibáñez M (eds) Evolutionary computation in combinatorial optimization. Springer, Cham, pp 146–157

Lipinski P, Michalak K (2019) Deriving knowledge from local optima networks for evolutionary optimization in inventory routing problem. In: Proceedings of the Genetic and Evolutionary Computation Conference Companion, ACM, New York, NY, USA, GECCO '19, pp 1551–1558

Liu SC, Chen JR (2011) A heuristic method for the inventory routing and pricing problem in a supply chain. Expert Syst Appl 38(3):1447–1456

Maghfiroh MFN, Redi AANP (2022) Tabu search heuristic for inventory routing problem with stochastic demand and time windows. J Sistem dan Manajemen Ind 6(2):111–120

Mahjoob M, Fazeli SS, Milanlouei S, Tavassoli LS, Mirmozaffari M (2022) A modified adaptive genetic algorithm for multi-product multi-period inventory routing problem. Sustain Oper Comput 3:1–9

Malladi KT, Sowlati T (2018) Sustainability aspects in inventory routing problem: A review of new trends in the literature. J Clean Prod 197:804–814

Michalak K (2021a) Feasibility-preserving genetic operators for hybrid algorithms using TSP solvers for the inventory routing problem. Procedia Computer Science 192:1451–1460, knowledge-Based and Intelligent Information & Engineering Systems: Proceedings of the 25th International Conference KES2021

Michalak K (2021b) Generating hard inventory routing problem instances using evolutionary algorithms. In: Proceedings of the genetic and evolutionary computation conference, Association for Computing Machinery, New York, NY, USA, GECCO '21, pp 243–251

Mirzaei S, Seifi A (2015) Considering lost sale in inventory routing problems for perishable goods. Comput Ind Eng 87:213–227

Mousavi R, Bashiri M, Nikzad E (2022) Stochastic production routing problem for perishable products: modeling and a solution algorithm. Comput Oper Res 142:105725

Mousavi SM, Niaki STA, Bahreininejad A, Musa SN (2014) Multi-item multiperiodic inventory control problem with variable demand and discounts: a particle swarm optimization algorithm. Sci World J 2014:136047

Popović D, Vidović M, Radivojević G (2012) Variable neighborhood search heuristic for the inventory routing problem in fuel delivery. Expert Syst Appl 39(18):13390–13398

Rabbani M, Mokarrari KR, Akbarian-saravi N (2021) A multi-objective location inventory routing problem with pricing decisions in a sustainable waste management system. Sustain Cities Soc 75:103319

Rey D, Neuhäuser M (2011) Wilcoxon-signed-rank test. In: Lovric M (ed) International encyclopedia of statistical science. Springer, Heidelberg, pp 1658–1659

Roldán RF, Basagoiti R, Coelho LC (2017) A survey on the inventory-routing problem with stochastic lead times and demands. J Appl Log 24:15–24

Shaabani H, Kamalabadi IN (2016) An efficient population-based simulated annealing algorithm for the multi-product multi-retailer perishable inventory routing problem. Comput Ind Eng 99:189–201

Sifaleras A, Konstantaras I (2020) A survey on variable neighborhood search methods for supply network inventory. Springer Proc Math Stat 315:71–82

Skalnes J, Andersson H, Desaulniers G, Stalhane M (2022) An improved formulation for the inventory routing problem with time-varying demands. Eur J Oper Res 302:1189

Tatsis VA, Parsopoulos KE, Skouri K, Konstantaras I (2013) An ant-based optimization approach for inventory routing. In: Emmerich M, Deutz A, Schuetze O, Bäck T, Tantar E, Tantar AA, Moral PD, Legrand P, Bouvry P, Coello CA (eds) EVOLVE: a bridge between probability, set oriented numerics, and evolutionary computation IV. Springer, Heidelberg, pp 107–121

Timajchi A, Mirzapour Al-e-Hashem SM, Rekik Y (2019) Inventory routing problem for hazardous and deteriorating items in the presence of accident risk with transshipment option. International Journal of Production Economics 209:302–315, the Proceedings of the 19th International Symposium on Inventories

Touzout FA, Ladier AL, Hadj-Hamou K (2022) An assign-and-route matheuristic for the time-dependent inventory routing problem. Eur J Oper Res 300(3):1081–1097

Vadseth ST, Andersson H, Stalhane M (2021) An iterative matheuristic for the inventory routing problem. Comput Oper Res 131:105262

Wang Z, Cheng S, Peng H (2019) Solve the IRP problem with an improved PSO. In: Peng H, Deng C, Wu Z, Liu Y (eds) Computational intelligence and intelligent systems. Springer, Singapore, pp 3–16

William Cook (2020) Concorde TSP solver. http://www.math.uwaterloo.ca/tsp/concorde.html, online: Accessed 19 May 2022

Wu W, Zhou W, Lin Y, Xie Y, Jin W (2021) A hybrid metaheuristic algorithm for location inventory routing problem with time windows and fuel consumption. Expert Syst Appl 166:114034

Yu VF, Widjaja AT, Gunawan A, Vansteenwegen P (2022) The multi-vehicle cyclic inventory routing problem: formulation and a metaheuristic approach. Comput Ind Eng 157:107320